分类号_____ 学号_____

学校代码_____ 密级_____

华中科技大学

硕士学位论文

基于 Web 的增材制造预处理平台设计与实现

学位申请人：张鹏

学科专业 ：计算机技术

指导教师 ：李国宽副教授

答辩日期 ：2021 年 5 月 23 日

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Professional Master Degree

Design and Implementation of Additive Manufacturing Pretreatment Platform Based on Web

Candidate : **Peng Zhang**

Major : **Computer Technology**

Supervisor : **Assoc.Prof. Guokuan Li**

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

June, 2021

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　在_____年解密后适用本授权书。
　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘 要

增材制造（Additive Manufacturing, AM）俗称 3D 打印，是一种通过材料逐层堆积的方式来构造物体、满足生产生活各领域个性化和定制化需求的制造技术。对模型数据进行预处理是增材制造过程中的重要环节。目前模型数据预处理软件多为客户端离线程序，不仅面临着算法被破解与数据丢失的风险，还有频繁更新和多端适配等问题；而随着 Web3.0 时代的到来，浏览器引擎的处理速度飞速提升，实时三维数据渲染的方案越来越完善，Web 应用可以与增材制造预处理流程有效地结合，建立起一个实用、便捷的增材制造数据预处理平台。

首先分析了增材制造用户对于预处理软件的需求，再根据需求设计了基于 Web 的增材制造预处理平台的系统框架和技术框架。系统框架包含了注册登录、用户模型管理、模型交互、模型数据预处理等功能模块。技术框架采用 Vue、Express.js、MongoDB、Three.js、Ant Design Vue 作为平台实现的底层支撑；然后阐述系统中的数据结构，根据 JavaScript 语言特性对模型处理过程中的冗余去除、拓扑重建、分层切片、轨迹填充等算法进行优化；并针对三维图层显隐控制的问题设计出基于递归的映射树构建方法。采用前后端分离的开发模式实现了对系统用户和模型数据的管理，以及三维模型文件的导入导出、分层切片、轨迹规划、GCode 生成等功能。最后，将系统部署在云服务器上。

对系统进行测试和验证的结果表明，增材制造数据预处理流程可以与 Web 应用良好地结合，系统交互界面合理，不存在明显功能缺陷，基本上可以满足增材制造数据预处理的需求，达到了预期的开发目标。

关键词：增材制造；预处理；Web 开发；STL 模型

Abstract

Additive Manufacturing (AM), commonly known as 3D printing, is a kind of manufacturing technology that uses materials to build objects layer by layer to meet the needs of all areas of production and life. Preprocessing model data is an important step in additive manufacturing process. At present, model data preprocessing software is mostly client-side off-line programs, which not only faces the risk of algorithm cracking and data loss, but also frequently updates and multi-terminal adaptation. With the advent of Web3.0 era, the processing speed of browser engine is rapidly improved, and the real-time three-dimensional data rendering scheme is becoming more and more perfect. Web application can be effectively combined with additive manufacturing pretreatment process to establish a practical and convenient additive manufacturing data pretreatment platform.

Firstly, the requirements of additive manufacturing users for preprocessing software are analyzed, and then the system framework and technical framework of additive manufacturing preprocessing platform based on web are designed according to the requirements. The system framework includes registration and login, user model management, model interaction, model data preprocessing and other functional modules. Vue, express. JS, mongodb, three. JS and Ant Design Vue are used as the underlying support of the platform; Then, the data structure of the system is described, and the redundancy removal, topology reconstruction, layered slicing, trajectory filling and other algorithms are optimized according to the characteristics of JavaScript language; Aiming at the problem of 3D layer hidden control, a recursive mapping tree construction method is designed. The development mode of front-end and back-end separation is adopted to realize the management of system users and model data, as well as the import and export of 3D model files, layered slicing, trajectory planning, gcode generation and other functions. Finally, the system is deployed on the cloud server.

The test results of the system show that the process of additive manufacturing data preprocessing can be well combined with Web application, the system interface is reasonable, there is no obvious functional defects, basically can meet the requirements of

additive manufacturing data preprocessing and achieve the expected development goals.

Key words: Additive Manufacturing; Pretreatment; Web development; STL model

目 录

1. 绪论

1.1 课题背景及研究意义

增材制造又被称为 3D 打印、积层制造，是一种以数字模型文件为基础，运用金属或塑料等可粘合材料，通过逐层打印的方式来构造物体的技术，最早提出于 20 世纪 80 年代[1,2]。增材制造的全流程会经过离散和堆积处理：离散是使用一系列预设高度的平面对数据模型进行切片，得到每一层平面的轮廓；堆积是根据不同的工艺方式，计算出每个轮廓的填充策略，生成每层的加工路径，再将加工路径翻译成打印机器可执行的加工代码，供机器进行实际生产[3]。

在增材制造过程中对模型数据的预处理十分重要，它包含模型导入、模型拓扑重建、模型切片、扫描路径生成等步骤，不同的参数以及计算算法都将直接影响到产品的质量和精度。市面上针对增材制造的数据处理软件主要基于客户端，一般分为国外开源的计算软件和打印机生产商提供的软件两种。前者需要下载安装包，后者则是将国外开源软件进行自定义的封装。对于普通用户而言，寻找安装包以及经常更新软件耗时且繁琐；对于专业用户而言，由于不同软件的算法不一致，定制化处理需求难以得到满足；对于打印机生产商而言，二次开发实施难度大，且存在代码侵权和软件冗余等问题。由此可见，客户端形态的预处理软件呈现出种种弊端，亟需新的架构模式来解决。

二十一世纪以来，互联网行业飞速发展，正逐步融入制造业的生产与商业环节中，引发全球工业产业技术升级。浏览器应用作为互联网行业发展中的重要一环，具有免安装、跨平台、无感更新、交互丰富等特性，数量逐年倍增。许多客户端软件开始逐步开发 Web 端应用，比如 Web CAD，Web Photoshop 和在线视频剪辑等。所以研究如何在 Web 实现增材制造预处理流程，使得应用操作精简化、计算统一化、数据共享化、功能平台化，具有重要的现实意义。

1.2 国内外研究现状

1.2.1 增材制造技术研究现状

增材制造与传统制造工业相比，最主要的区别是它不需要模具，而是利用计算机建模数据，就可以通过逐层累加的方式进行填充打印。目前最常见的增材制造技术是熔融沉积技术（Fused Deposition Modeling，FDM），该技术适用于大多数复杂模型的制造，被广泛应用于建筑模型构建、航空机械零件设计、医疗器械制备等领域[4-8]。因此，后续系统都是基于 FDM 来设计的。

一般基于 FDM 的增材制造主要流程是，先通过建模软件得到满足需求的模型文件，将得到的模型文件通过专业软件进行分层切片处理，根据切片得到的轮廓判定轮廓之间的内外关系；再根据轮廓计算出不同的包络区域进行路径规划填充，最后将规划路径转为打印机可执行的 G 代码；G 代码生成后，根据不同工艺手法去进行实际打印，以熔融沉积工艺过程为例：打印机读取 GCode，加热打印喷头，将融化的材料匀速挤出，沿着每个切片平面方向进行均匀涂抹，直至不断调整高度填充整个模型，生成最终的打印形件。常见 FDM 型 3D 打印机结构如图 1-1。

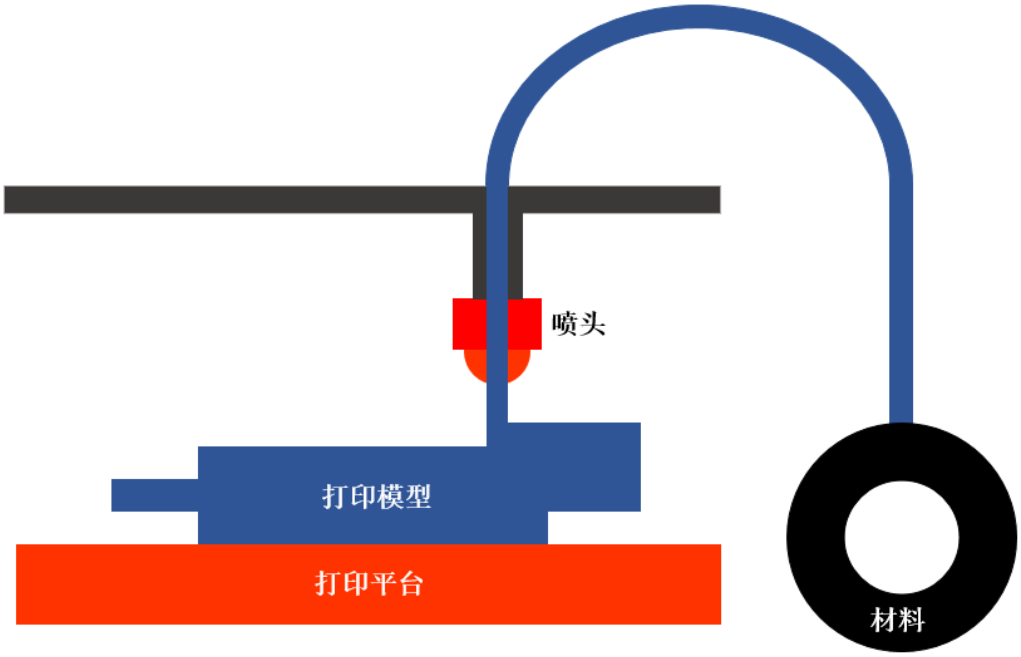


图 1-1 常见 FDM 型 3D 打印机结构示意图

根据上述流程可知增材制造预处理一般包含以下五个关键要素：

1. 三维模型建模：三维模型制作方式有很多种，比如人工建模、扫描建模，根据不同建模方式所选择的软件也不同，如：CAD、3DMax、Zbrush、SketchUp 等[9]。

2. STL 文件：常见的三维文件格式有 ABC（动画模型文件格式）、glTF（动画模型文件格式）、FBX（3D 文件格式）、BVH（人体特征动画文件格式）、OBJ（标准 3D 模型文件格式）、DAE（纯文本模型格式）、STL（三角网格文件格式）等，STL 由于格式简单，易于转化输出，因此被广泛用于 3D 打印和计算机辅助制造领域。

3. 分层切片：STL 模型在结构上是由一个个三角面片组成，进行分层切片时，需要将同一层相邻的面片顺序存储在每个轮廓的数据结构中，分的层数越多，模型精度越高，但耗费的计算时间与计算空间也越大。

4. 路径规划：分层处理得到轮廓数据集后，需要找出内外轮廓关系，划分不同的填充区域，还要根据打印机特性设置不同的填充方式，保证合理地生成打印轨迹。

5. G 代码生成：根据不同的工艺去配置 GCode 相关参数，然后使用 GCode 去操作配套打印机进行填充制造，直至最终形成三维实体器件。

在预处理的过程中，数据信息流的传递方式如图 1-2。

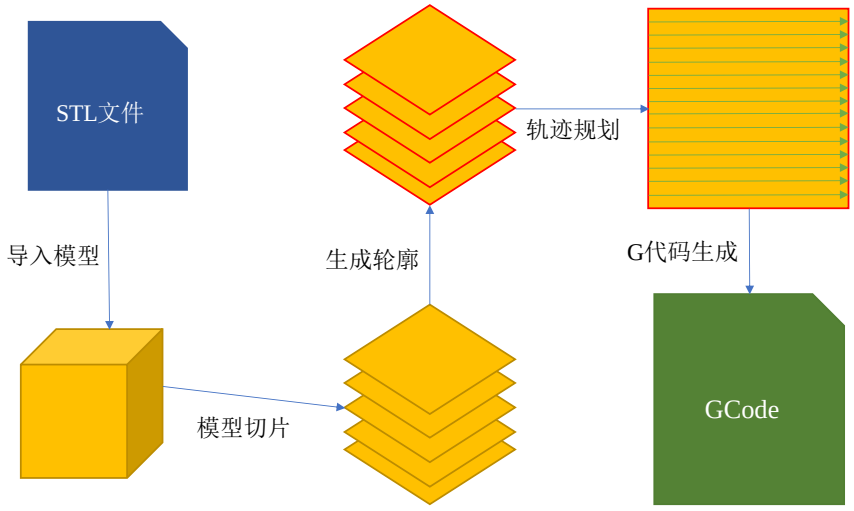


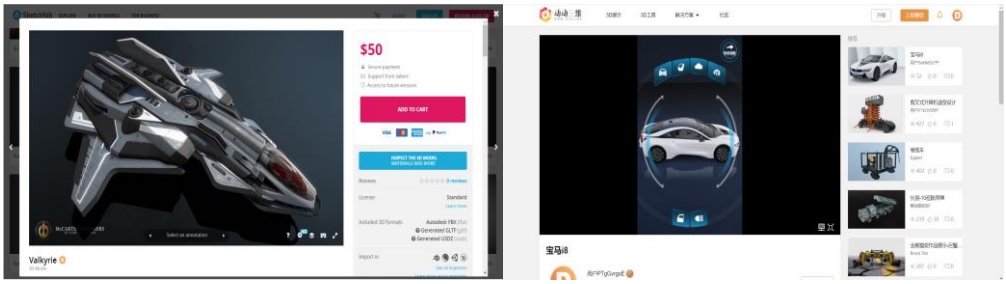
图 1-2 3D 预处理过程数据信息流示意图

预处理过程要用到专业的三维数据处理软件，大致可分为面向工业产品设计制造的计算机辅助设计系统（Computer-Aided-Design），用于电影、动画等行业的计算机图形系统（Graphic）以及面向增材制造应用的三维模型设计系统[10]这三类。从 1960 年代的 SketchPad[11]系统开始，到 1970 年代 Rochester 大学的 Herb Voelcker 等人研

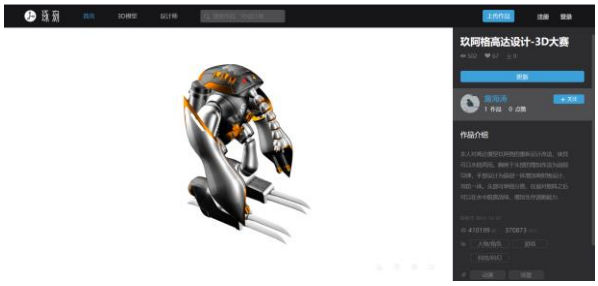
发出 PADL 建模器[12-14]，再到 1980 年代剑桥大学的 Ian Braid 和 Charles Lang 等人提出 BRep（Boundary Representation）建模技术[15-17]，大大促进 ACIS 等建模器的诞生。此后 SolidWorks 公司和 AutoDesk 公司分别于 1995 年和 1999 年发布 SolidWorks 设计系统和 Inventor 设计系统，成为计算机三维辅助设计软件的首要选择。

随着 3D 打印技术的发展，市场上开始出现专业的增材制造处理软件，比如：Simplify3D、Cura、MakerBot Print、Ultimaker、CraftWare，Slic3r 等[18]。其中 Simplify 3D 是商用切片软件，源码保密，售价高；Slice3r 是开源软件，兼容 FDM 设备、DLP 和 LCD 打印机，能将分层切片保存为 SVG；Cura 也是免费开源软件，因其丰富的参数设置、广泛的机型兼容性，受到众多 3D 打印用户的喜爱。上述软件虽可以满足预处理需求，但从软件生态角度而言只是一个高效的工具，无法做到在线用户信息与模型数据管理，也无法构成完整的平台服务体系。

国内 3D 打印软件研究始于 90 年代，不同机构根据成形原理、打印材料和加工工艺等方面开发了一些比较成形的软件[19]。由于国内研究起步晚，面向高端制造业，产品价格极高。对于大众 3D 打印市场，只有数十家企业在从事经济型 3D 打印机的制造，且配套软件都是将国外开源软件做汉化处理，核心算法难以改进，易涉及知识产权问题。此外，软件的推广模式也存在弊端。



（a）Sketchfab　　（b）动动三维



（c）琢刻

图 1-3 Sketchfab[20]、动动三维[21]、琢刻[22]Web 应用界面

完善的模型数据是增材制造预处理的基石。在国外 2018 年 6 月 Sketchfab（Web 上的 3D 和 AR 数据管理平台）上线，在国内阿里巴巴于 2018 年 5 月推出动动三维（全球首款 3D/AR 交互内容在线工具）和琢刻（在线上传、分享和发现 3D 模型），上述几款 Web 应用界面如图 1-3。因此将模型管理与增材制造预处理相结合，构建出在增材制造细分领域的 Web 应用平台，会打造和延展 3D 内容生态链。

1.2.2 Web 技术研究现状

自 1989 年蒂姆·伯纳斯·李（Tim Berners-Lee）发明万维网以来[23]，Web 应用数量开始爆炸式增长。Web 页面通常由 HTML（Hyper Text Markup Language）[24]、CSS（Cascading Style Sheets）[25,26]、JS（JavaScript）[27,28]三部分组成。早期的 Web 页面只能对图片、文字等静态数据进行展示。随着网站功能愈加复杂，后端混编前端代码来生成页面模板的形式变得难以维护，于是在 2004 年提出 Ajax（Asynchronous Javascript And XML）[29]，它是一种创建交互式、快速动态网页应用，无需重新加载整个页面，就能够更新部分网页的开发技术，降低了前后端代码的耦合性。随着前端数据交互变得便捷，网页逐渐从信息门户网站过渡到社区论坛和电商平台等多数据交互的应用。2005 年 Flash 插件问世[30]，由于可以在浏览器中播放动画，Flash 引爆了整个流媒体行业，网页游戏浪潮席卷全球，于 2010 年达到巅峰，直至 2015 年 Flash 暴露出重大安全问题，才渐渐地被 HTML5 和 CSS3[31,32]所替代。

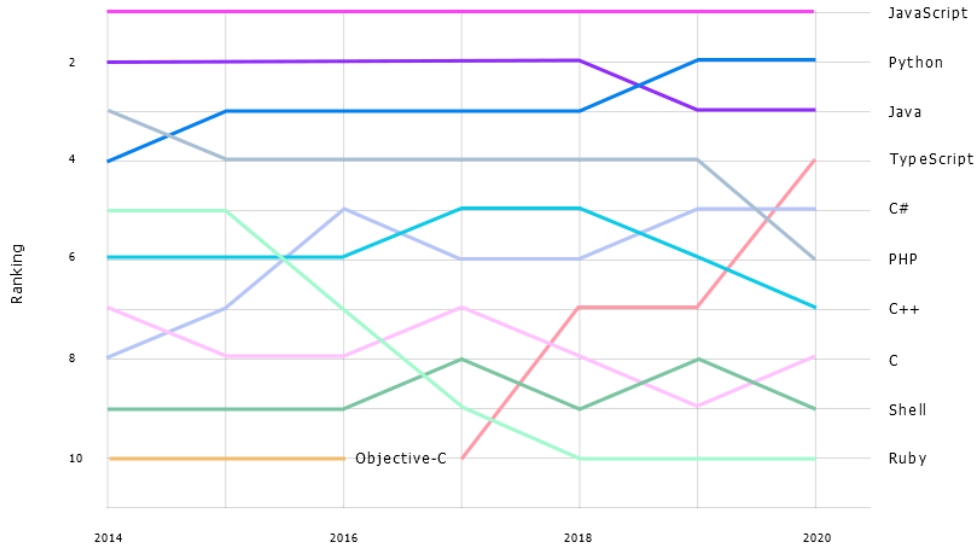


图 1-4 Github 近六年最受欢迎的编程语言排名

Javascript 作为 Web 开发的核心语言，从 Github 近六年最受欢迎的编程语言如图 1-4 可知， Web 技术同比与其它开发技术，发展更为迅猛。JavaScript 的开发框架最早是 2006 年发布的 jQuery，核心设计理念为写更少的代码、做更多的事情（Write Less Do More）[33]。CSS 框架库最早是 Twitter 于 2011 年开源的 GUI 框架 Bootstrap，对 CSS 做了不同浏览器的兼容处理[34,35]。随着语言标准的更新，Web 应用的运行环境也在升级。目前浏览器市场占有率最高的是 Google 公司在 2008 年开发的网页浏览器 Google Chrome。Google Chrome 基于 webkit 内核[36]。由于良好的设计标准和市场反应，便捷的调试工具以及丰富的插件市场，浏览器快速迭代，从而受到了广大 Web 程序开发者的喜爱。Google Chrome 内置 V8 引擎[37]，显著提升 Javascript 的运行速度，满足了高性能 Web 应用的需求。此外，Node.js 作为 Javascript 的运行环境，通过事件驱动、非阻塞式 I/O 的机制，确保应用在高并发下流畅运行[38-40]。同时 Node.js 内置 NPM 包管理器[41]，大量工具资源得以共享，推动着前端行业增量式发展。

近几年由于底层技术的换代，开发模式也发生较大变化。早期网站的服务模式是先发送一个 http 请求，然后服务器对请求做处理，最后将静态资源返回到页面中[42,43]。因此静态资源和业务代码统一部署在同一台服务器上，整个流程如图 1-5。这种模式下，前端开发流程是将 UI 设计师提供的原型图生成静态页面交付于后端工程师，再让后者加入逻辑代码，导致前后端工作极度耦合、相互依赖严重、开发效率低、代码难以维护，一旦设计改变就需要重新走一遍流程。直到 2013 年，前后端分离思想出现。在基于 Node.js 服务器的开发模式下，前端负责 View 层和 Controller 层，后端负责 Model 层和 Service 层，浏览器不再直接请求服务端的 API，而是先去请求 Node.js，由 Node.js 对服务端的 API 发送 HTTP 请求，Node.js 收到服务端返回的 JSON 格式数据再去渲染 HTML 页面[44]。前后端分离不仅将开发人员的职责明确区分开来，降低开发流程耦合度，还提升了适配性。

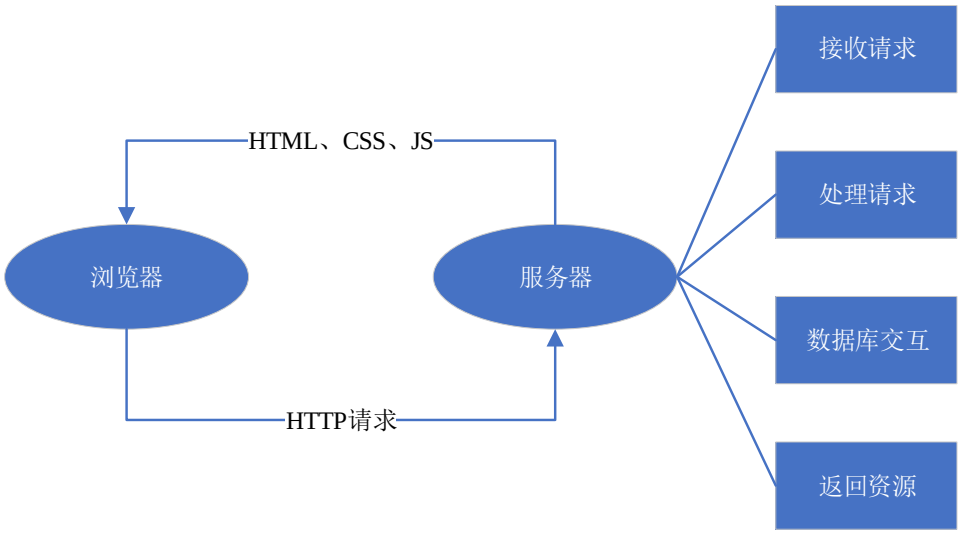


图 1-5 早期网站服务流程

随着浏览器 GPU 调用逐步支持和 3D 编程需求与日俱增，WebGL 生态下的应用有着广阔的发展空间。WebGL 是一种用于 Web 的标准 3D 绘图协议，基于 OpenGL ES2.0 的 JavaScript API 进行开发。在 1992 年发布，目前由 The Khronos Group 管理。它是一种基于渲染管线、独立于硬件，且具有客户端-服务器结构化的 API[45,46]。WebGL 的问世，使得网页应用可以开拓出更多可能性，例如 3D 页游、数据可视化、虚拟现实等。

Web 技术从诞生以来经过近二三十年的发展，发生了天翻地覆的变化，从最初只能制作一个简单的静态文本页面，到如今可以涵盖 Web OS、AR、VR 和人工智能等前沿科技领域的强大技术。截止到 2020 年底，全球网站数量已经突破十亿，尤其是近十年来，网页数量呈几何级数增长，广泛应用于各个行业。许多基于客户端的程序都逐渐向 Web 端扩展，比如 Web Photoshop、AutoCAD Web App、Tim 在线文档等产品。Web 化的好处不仅是无感知更新，即开即用，还能将计算云服务化、数据云存储化、服务云平台化。Web 应用程序曾经最为诟病的问题是执行效率远低于客户端程序，但是随着底层硬件性能不断被提升、技术瓶颈不断被突破、浏览器厂商与行业标准不断进步，Web 应用程序的执行效率已快速提升。客户端程序向 Web 端扩展，是未来互联网应用发展的必然趋势。

1.3 本文研究内容及安排

本文通过分析增材制造预处理流程和调研预处理相关软件，设计并实现基于Web的增材制造预处理平台。平台包括模型处理算法设计、GCode生成、交互式界面、数据库设计、服务端设计五个模块。对WebGL技术和Three.js框架进行分析，并基于开放的底层API对系统中模型展示与交互操作进行算法设计、编码实现。确定平台应用的技术选型，完成Web前端、NodeJS后端以及数据库的搭建，并对整个应用架构进行组合设计。研究分析增材制造预处理过程中涉及到的几大算法，详细阐述了每个算法的具体实现流程，再根据实际Web应用开发的数据结构，对算法进行优化调整，并用实际模型对算法进行测试。将应用程序部署在服务器上，并对系统进行测试分析。

本论文的结构安排如下：

第一章绪论，主要介绍本文课题的研究背景与研究意义、 增材制造技术的研究现状以及Web技术的研究现状。

第二章增材制造预处理 Web 平台的原理及关键技术，介绍增材制造预处理相关算法和Web应用全栈化开发相关技术。

第三章系统需求分析及概要设计，对系统进行整体概述，明确系统的功能性需求与非功能性需求，得到系统用户的用例分析，再从界面、功能、平台角度出发设计出系统整体架构，最后对系统功能进行模块设计。

第四章核心算法模块设计，对平台中所涉及到的关键算法进行分析研究，并基于Web开发的特性和可视化需求来改进对应算法，使其满足在线实时处理需求。

第五章增材制造预处理 Web 平台实现与测试，对开发环境搭建、前端实现、模型交互实现、数据库实现和服务端实现这五部分的具体实施细节进行阐述，最后对系统进行测试与结果分析。

第六章总结与展望，总结本文研究的主要内容，并指出系统研究实现中遇到的问题和不足，最后对基于Web的增材制造预处理平台进行了展望。

2. 增材制造预处理 Web 平台的原理及关键技术

2.1 增材制造预处理相关技术

2.1.1 STL 模型简介

立体光造型文件（Stereo Lithographic File，简称 STL 文件）原本用于立体光刻电脑辅助设计，现被广泛应用于快速成型、3D 打印和电脑辅助制造等领域。其基本思想是利用离散的微小三角面片来组成各种三维模型的曲面，因格式简单，易于生成，可以被许多计算机辅助设计软件所使用[47]。每个 STL 文件都是由一组笛卡尔坐标系下无序无规则的三角形顶点和法向量组成的，STL 坐标必须是正数，不包含尺度信息，并且计量单位是任意的。如图 2-1 为一个 STL 格式文件的球体。

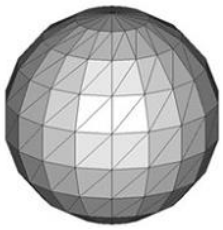


图 2-1 STL 模型

STL 文件格式分为两种：一种是 ASCII 格式，另一种是二进制格式。具体的文件结构如下：

1. ASCII 格式：

solid StlFileName // 文件路径及文件名

facet normal nx ny nz // 三角面片法向量的 3 个分量值

outer loop

vertex x y z // 三角面片第一个顶点坐标

vertex x y z // 三角面片第二个顶点坐标

vertex x y z // 三角面片第三个顶点坐标

end loop

end facet // 完成一个三角面片定义

.....// 处理其他三角面片

endsolid StlFileName // 整个 STL 文件定义结束

2. 二进制格式：

UINT8[80] Header // 使用 80 个字节存储文件名

UINT32 Number of triangles // 模型的三角面片数量

// 每个三角面片使用 50 字节存储数据，具体规则如下：

REAL32[3] Normal vector // 每 4 个字节浮点数存储法线矢量的一个分量坐标

REAL32[3] Vertex1 // 12 个字节存储 1 个顶点坐标

REAL32[3] Vertex2

REAL32[3] Vertex3

UINT16 Attribute byte count end // 2 个字节存储三角面片属性

通过对 STL 文件存储格式的分析，确定以 STL 文件作为增材制造预处理平台的标准模型文件。

2.1.2 STL 模型冗余去除与拓扑重建

在 STL 模型文件结构中，每个面片结构都包含各顶点的信息，通常情况下每个顶点都会被多个三角面片所共享，这就会出现数据的冗余[48]。将冗余数据进行下一步处理，必然会导致重复计算，浪费内存空间和增加计算时间。

以一个大三角面片为例，其中包含三个小三角面片，如图 2-2，p_i（i=1，2…4）为模型顶点，e_i（i=1，2…4）为模型边，f_i（i=1，2…4）为模型面。从面片数据组成元素可知，每个顶点和每条边在原始数据中都被重复存储多次，当模型数据复杂时，三角面片数量巨大，冗余数据点增多，导致搜索效率低下。一般处理方式是根据实际情况在读取模型数据时转换存储格式，尽量保证相同的数据只存储一次。

拓扑重建[49,50]是为了建立三角面片的拓扑结构，STL 模型数据格式是一个个无规律的三角面片而分层切片需要使用到面片之间的空间结构信息，如果只对无序数据搜寻轮廓轨迹，那每次循环都会遍历所有面片，但实际上与切平面相交的面片只占少数，使用空间结构信息就可以将大量无效数据剔除，提升分层切片效率。并且拓重建与冗余数据去除可以同步进行，基本思想是牺牲空间换取时间的散列法。

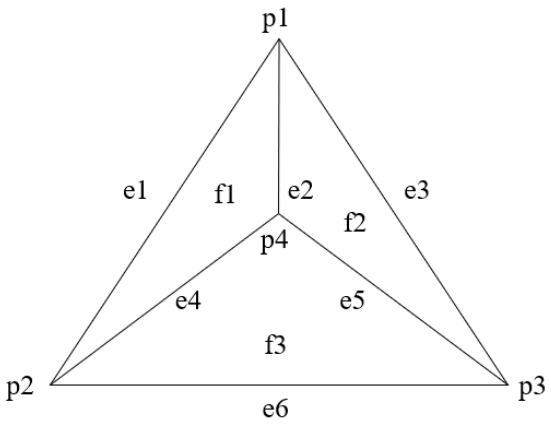


图 2-2 三角面片组成元素示例

2.1.3 STL 模型分层切片

分层切片目的是将模型数据离散化，从而得到每层的二维平面数据。切片过程根据起始高度、终止高度、切片层厚等参数来对网格模型进行平面截取，对截平面与截平面所处高度下的三角面片求交点，再将所求交点顺次连接成一个封闭多边形轮廓。目前基于 STL 文件的分层切片算法，主要集中在等厚分层切片算法，一般分为三类[51]：

1. 直接求交线法：根据给定切平面高度 Z 去遍历所有面片数据，求出与该平面相交的线段，再依次连接交线段形成封闭轮廓，得到模型在高度 Z 下的多边形截面轮廓。此算法流程简单，易实现，但对于复杂模型数据，尤其是包含大量三角面片数据的模型而言，该算法耗时久，效率低下。

2. 基于模型拓扑结构的切片算法：由于分层切片是寻找特定高度下的相交轮廓，而组成轮廓的面片之间就会有空间拓扑信息，因此可以提前构建空间拓扑结构，利用面片间的几何关系来生成相交轮廓，减少面片的遍历次数。此算法效率有较大提升，但是提前存储空间拓扑结构会占用大量内存。

3. 基于模型连续性的切片算法：首先将三角面片建立集合，在分层过程中，动态更新与当前切面高度相交的面片数据，行进到下一层高度时，先从面片集合里删去不相交的面片，同时将与该高度相交的面片加入到集合中，建立局部三角面片拓扑信息，然后求交线，获得轮廓线段，直至分层结束。该算法使用动态更新局部拓扑结构，减少了存储全局拓扑信息的空间，然而算法数据结构较为复杂，实现难度较大。

2.1.4 轨迹规划

切片完成得到多边形轮廓后，需要根据用户设置的线宽、填充密度等参数对封闭轮廓线区域生成轨迹填充路径。路径类型主要分为轮廓路径和填充路径两种。使用轮廓路径填充时，打印机每次路径的起点就是终点，因为轮廓路径填充只是将轮廓进行了等距向内或向外的偏置，不断沿着偏置的轨迹打印，直至填充满轮廓内部区域；路径填充则是将特殊的线段按照一定的规律对内部区域进行扫描，所以路径可以为任意位置，每次打印完该线段后的位置跟下一个线段的起始位置都可能不同。

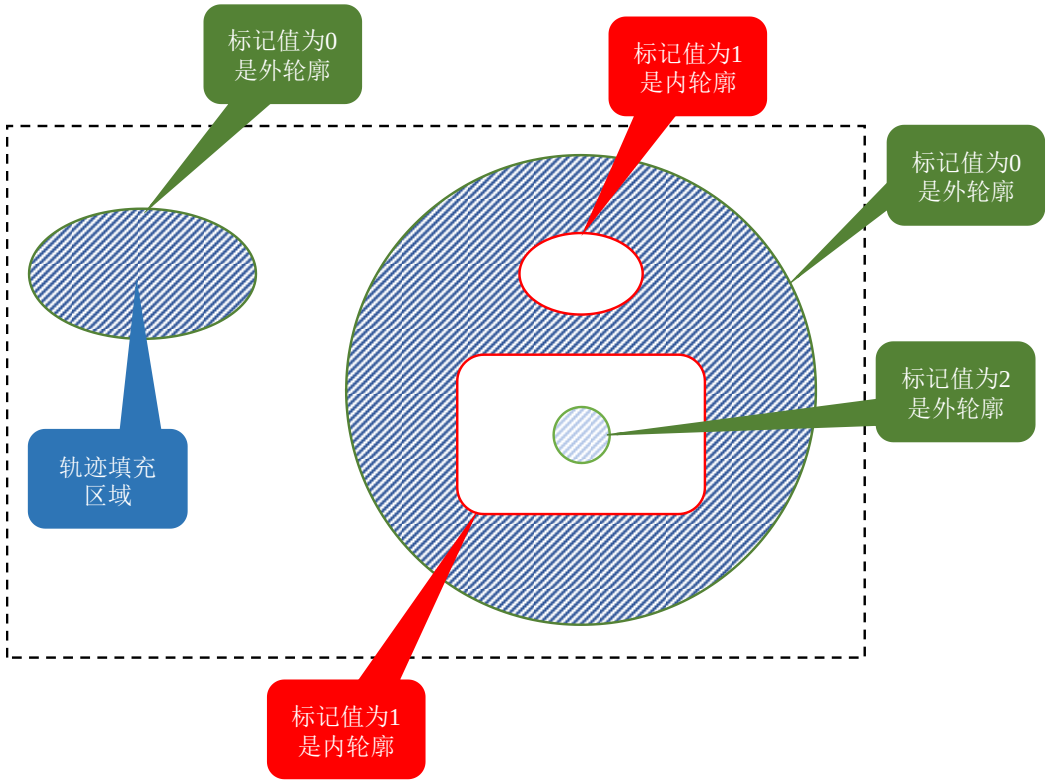


图 2-3 复杂模型内外轮廓示意图

对于复杂轮廓或者模型中间有孔洞的三维模型，切片截面内部存在非打印区域，会形成内外轮廓[52]。多轮廓图形的区域几何关系通常是包含与分离，在执行扫描线多边形填充算法前还需要对复杂轮廓进行内外轮廓判定，确定需要被填充的区域。区分内外轮廓的步骤一般为：首先计算出每个轮廓在笛卡尔坐标系中 x 和 y 的正负轴方向上的最值，即 xMin、xMax、yMin、yMax，根据这四个值的关系得出轮廓之间的位置关系；再给每个轮廓初始化一个等于 0 的标记值，两两进行对比，如果一个轮

廓被另一个轮廓包含，则给被包含轮廓的标记值加 1，记录该轮廓被包含的次数。由几何关系可知，当轮廓的标记值为奇数时，该轮廓为内轮廓；当标记值为偶数时，该轮廓为外轮廓。根据这个算法，可以很快得出轮廓之间的内外关系，最后计算出需要进行轨迹规划处理的区域。比如一些复杂模型的内外轮廓区分如图 2-3，图中的绿色线条表示外轮廓线，红色线条表示内轮廓线，蓝色线条表示需要轨迹填充的区域。

2.1.5 GCode 格式简介

打印路径数据确定后，按照 Three.js 的绘制数据格式进行存储，而要让打印机执行打印指令，还需要将计算出来的路径数据转化为 3D 打印机可识别的电路代码。不同的打印机有不同型号的运动控制器，因此必须转化为一种标准代码，也就是 GCode。GCode 原本是用来控制数控机床加工模型的一种语言，可以控制机床的切削工具按照指定的路径对零件进行切削剪裁，形成所需的零件[53,54]。

GCode 语言格式明确，规则完善，代码可读性高，用户能便捷地查看到加工路径的位置坐标，所以大部分打印机都使用 GCode 作为运动控制器的控制代码。GCode 代码的格式由一个英文字母（A-Z）加一串数字（0-9）组成，字母 G 开头的指令用来控制打印机喷头运动，字母 M 开头的指令是一些辅助命令，字母 X 和 Y 开头的指令用来控制喷头在 X 轴、Y 轴上运动位置的增减，字母 E 开头的命令代表着喷头的出丝量，字母 F 开头的指令表示喷头移速。常用 G 代码指令含义如表 2-1。

表 2-1 常用 G 代码指令含义

代码	含义
Gddd	控制喷头运动
Mddd	辅助命令
Tddd	选择哪种打印头
Sddd	是否检查限位开关
Pddd	命令参数，例如时间
Xddd	坐标点的 X 轴位置
Yddd	坐标点的 Y 轴位置
Zddd	坐标点的 Z 轴位置
Eddd	喷头出丝的挤出量

表 2-2 常用 G 代码指令含义（续）

代码	含义
Fddd	打印喷头移动速度
Rddd	温度相关参数
Qddd	目前未使用参数
Iddd	目前未使用参数
Nddd	行号
*ddd	校验码，检查通讯错误

在打印机读取 GCode 命令前，需要对打印机进行初始化：打印头位置归零、对打印材料进行预处理、调整底部平台、启动温度装置、抬起打印头、打印头测试出料、打开制冷风扇冷却路径。每层打印开始前，要将打印头的位置移动至该层高度的 Z 坐标处，根据路径数据，每个点都是一行 GCode 指令，指令里包含了打印头移动速度、出料长度和宽度、坐标点位置等，打印头行进速度可以根据需要调整，空行程代码为 G0，否则为 G1。当前层打印结束后，关闭温度装置，打印头回到原点。

2.2 Web 应用开发相关技术

近几年 Web 应用开发技术栈百家争鸣，在基于前后端分离的模式下，主流前端框架有 Angular、React 和 Vue，服务端有 Java、PHP、Golang、Python 和 NodeJS，数据库有 MySQL、Oracle、Redis、MongoDB 等。鉴于系统的计算模块使用本机计算资源，模型数据是 JSON 格式的文档流结构，因此本文选择编程语言统一的轻量型框架，也就是 Vue、Ant Design Vue、Three.js、Express.js 和 MongoDB 对系统进行开发。

2.2.1 Vue 框架简介

Vue[55]是一套用于构建用户界面的渐进式框架，基于视图层逻辑进行开发，易于上手，便于与第三方库整合，有以下四个特点：

1. 前端 MVVM

MVVM 模式[56]是从 MVC 设计模式[57]演变而成，MVC 设计模式如图 2-4,分为 Model（数据模型层）、View（视图层）、Controller（控制层）。其中 M 表示业务模型，

C 表示控制器，V 表示用户界面。MVVM 模式如图 2-5，它们最大的差异是 MVVM 通过 ViewModel（视图模型）分离视图与模型，实现 View 与 Model 的自动同步，极大地方便开发者对 DOM 的操作，提升开发者的工作效率。

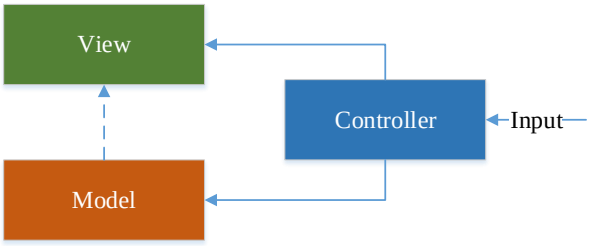


图 2-4 MVC 模式

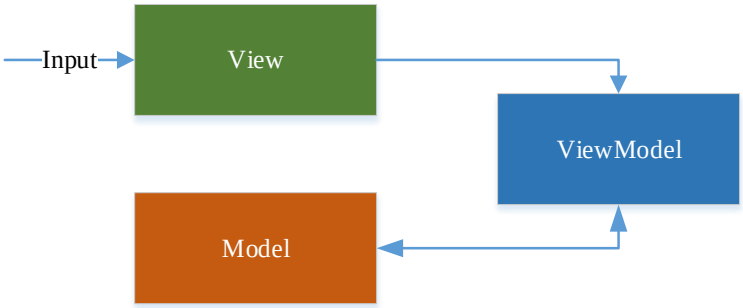


图 2-5 MVVM 模式

2. 虚拟 DOM

Vue 视图渲染过程使用的是虚拟 DOM 技术，内置渲染引擎的工作流程总共有 5 步：首先创建 DOM 树，然后创建 StyleRules，其次创建 Render 树，再布局 Layout，最后绘制 Painting。

3. 组件化开发

Vue 的组件化开发，支持将小型、独立和通常可复用的组件拿来构建大型应用，对组件成员都提供了相应的管理机制，方便开发者进行注册或实例化。

4. 服务器单独部署

在应用开发阶段，Vue 项目是在 Node.js 的环境下进行开发的，使用了 NPM、Webpack 为工具来部署 Web 服务，辅助开发人员对代码进行调试。

在实际生产阶段，将项目代码编译打包成静态的 HTML、JavaScript 以及 CSS 文件，然后部署到 Nginx 或者 Apache 服务器上完成前端应用部署。

2.2.2 Ant Design of Vue 简介

Ant Design of Vue 是基于 Ant Design 设计体系的 Vue UI 组件库，服务于企业级产品。主要有以下三大特点：一是使用企业级中后台产品的交互语言和视觉风格，具有较为良好的用户体验；二是开箱即用的高质量 Vue 组件，省去开发者大量重复封装的时间；三是共享 Ant Design of React 设计工具体系，降低开发者学习成本。

2.2.3 Three.js 简介

Web 端三维绘图离不开 WebGL 技术，该技术标准基于 JavaScript 和 OpenGL ES2.0，无需配置 Flash 等额外绘图渲染插件，直接使用浏览器环境操作复杂的三维结构和 3D 游戏。一般网页使用 HTML、JavaScript 和 CSS 呈现动态效果，而 WebGL 则是一种更为复杂的网页展示技术，借助系统显卡来对图形数据进行加速处理[58]。

Three.js 是目前 Github 上用来处理 Web 端 3D 程序技术生态中最为庞大的开源项目。Three.js 对 WebGl 的底层 API 做了二次封装，可以让用户按照一定的规律去绘制各种复杂的图形，并且提供许多场景交互接口和动画生成方法，大大地降低了三维模型处理程序的上手开发难度。

2.2.4 Express.js 简介

Express.js 是一个基于 Node.js 平台的快速、开放、极简的 Web 服务开发框架[59]。它提供了强大的 API，可以有效地实现路由、HTTP 请求、静态文件等功能，快速搭建定制化后端接口服务。

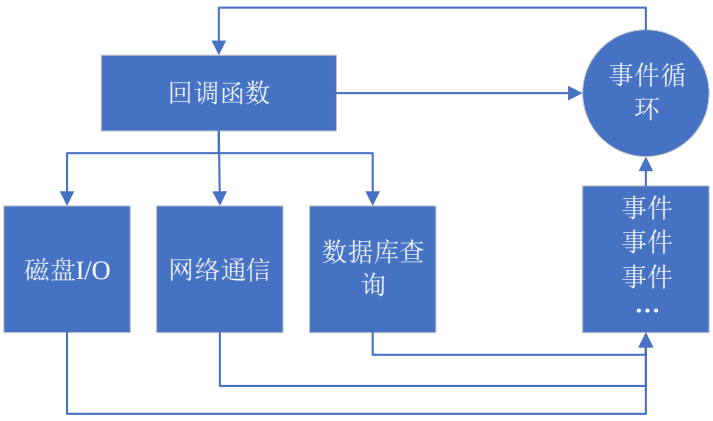


图 2-6 事件驱动模式

Node.js 具有以下三大特点：①单线程。当用户连接时，就触发一个内部事件，通过非阻塞 I/O、事件驱动机制在宏观上模拟并行机制。②非阻塞 I/O。当访问数据库时，非阻塞 I/O 机制将数据库返回结果后的处理代码放在回调函数中，保证主线程的代码继续执行。③事件驱动。当用户执行建立请求连接、提交数据等操作时，会触发相应事件，而在每一个时刻，只能执行一个事件的回调函数，但是在处理事件回调线程时，还能转去处理其它事件，其它事件处理完毕后还可以继续处理原事件的回调函数，这种处理机制被叫做“事件环”机制。事件驱动模式如图 2-6。

2.2.5 MongoDB 简介

随着 Web2.0 的兴起，非关系型数据库[60]成为热门研究领域。因为传统关系型数据库在面对大规模用户、海量数据以及高并发的 SNS（Social Networking Service）实时动态网站需求时，会有以下几种问题：①高并发需求出现性能瓶颈[61]。传统关系型数据库在面对每秒上万次查询还能承受，但是面对上万次数据库的读写请求时，磁盘 I/O 将无法支撑运行；②海量数据无法高效存储和访问，比如一个用户数据里面订阅了多个栏目，每个栏目里面又包含了不同种类的刊物，每个刊物里面又关联了作者，此时需要查阅该用户下有多少个服务的作者，关系型数据库通常会分库分表切分数据，但是遇到高并发请求，查询效率变得十分低下，并且易增加后期数据库维护和迁移的难度。③关系型数据库具有 ACID[62]特性，可是大部分网站中，许多数据操作都无法严格满足事务的特性，因此难以保证高负载下的数据一致性。

MongoDB 是一个面向集合（Collection）、模式自由（Schema-Less）、文档型（Document）的数据库。面向集合指的是数据被划分存储在数据集合里面；文档型指的是数据存储在文档中，每个集合可以包含无数个文档，文档本身就是一组键值对，键值对的键可以是任意数据类型，这种格式也被叫做 BSON（类似于 JSON 的二进制数据格式，两者都支持内嵌对象和数组）。MongoDB 数据库还有一个最大的优点，它是基于磁盘的数据库，面对海量数据的读写时仍然有高效率的 I/O。官方测试结果表明，当数据量大于 50GB 时，MongoDB 要比 MySQL 快将近 10 倍，

由于本文要存储 STL 模型数据，在 STL 模型数据被导入到 Web 端时是 JSON 格式，并且要与用户进行关联存储，所以文档式的数据库最符合系统需求，因此本文选

择 MongoDB 作为系统的数据库。

2.3 本章小结

本章主要介绍了基于 Web 的增材制造预处理平台所要用到的原理及关键技术。介绍了增材制造预处理过程中涉及到的数据结构，并详细阐述 STL 模型拓扑重建算法，切片算法，轨迹规划算法和 GCode 生成算法的原理，然后对比分析了该 Web 系统所用到的技术栈，包括前端 Vue 框架、前端 UI 库 Ant Design of Vue、Web3D 开发框架 Three.js、后台服务端框架 Express.js 和数据库 MongoDB，最后完成系统的整体架构选型。

3. 系统需求分析及概要设计

3.1 系统整体概述

目前市场上的增材制造预处理软件都是基于客户端，平台兼容性差、更新操作繁琐、二次开发难度大、软件同质化严重。对于复杂的模型处理需求，用户还要将多种软件进行组合使用。此外，用户本地存储着各种模型数据，这些数据存在着丢失的风险，并且随着模型数据量的增加，一些重复的模型会占用大量本地存储资源，不利于用户管理模型数据。基于 Web 的增材制造预处理平台能直接在网页中管理模型数据，且预处理算法的执行速度与模型展示效果都能满足用户需求，推动增材制造相关技术云服务化的进程。

鉴于上述原因，本文首要任务是：在平台上，用户可以管理模型数据并可以将模型数据共享，还能对模型进行基础三维展示、拓扑重建、分层切片、轨迹规划、GCode 生成以及数据日志记录。该平台适用于有增材制造需求的相关行业。

在确定需求时，主要通过与增材制造行业客户面谈，以及与相关技术从业人员进行讨论的方式来确定。结合多方观点和现实需求，对整体系统的应用场景，系统模块，细分功能进行设计，以流程图和功能模块图的形式展现，最后与技术人员讨论，确定出不同功能依赖的数据结构以及对应的实现技术。总体采用原型法进行设计，并使用敏捷开发推进整个项目。

因此，基于 Web 的增材制造预处理平台的主体需求为：

1. 用户数据管理；
2. 模型数据管理；
3. 模型展示和交互动画；
4. 增材制造预处理全流程；
5. 模型处理过程中的日志记录；
6. 实时查看页面性能。

3.2 需求分析

基于 Web 的增材制造预处理平台的核心工作流如下：

1. 注册系统用户，注册完成后登陆系统进入模型管理中心；
2. 用户上传本地模型到个人模型仓库内，并设置模型相关的参数，也可直接从所有模型库中选择其他用户模型放入我的模型库内；
3. 选择一个模型进入工作台，在工作台页面中可对模型进行放缩、回中、旋转、显隐坐标轴、显隐底部参考网格、显隐模型等展示操作；
4. 模型切片处理，再将结果渲染到画面上，更新图层数据，并加入日志记录；
5. 对模型进行轨迹规划处理，渲染画面，更新图层数据，并将记录加入日志；
6. 按照轨迹规划的路径数据生成相应的轨迹打印动画；
7. 生成 3D 打印机可识别的 GCode，并提供下载 GCode 文件到本地的功能。
8. 模型处理操作的追踪数据都记录于日志面板中。

具体流程如图 3-1。

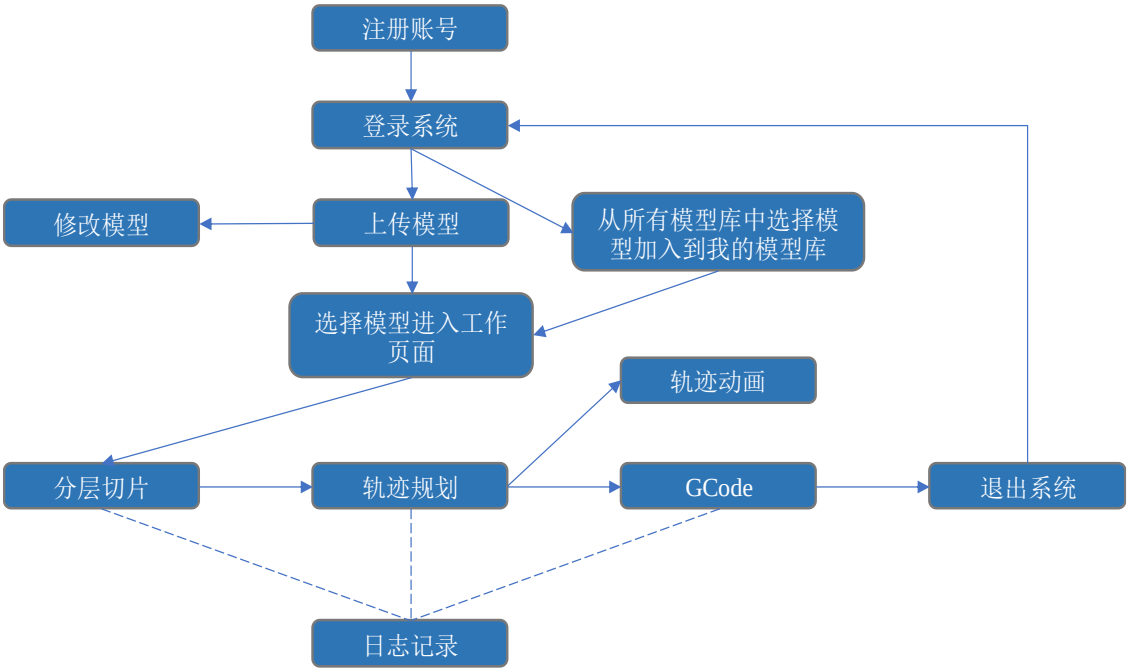


图 3-1 平台核心工作流

针对上述工作流进行系统的需求分析，将分为功能性需求和非功能性需求两种。

3.2.1 功能性需求

增材制造数据预处理平台给用户提供模型数据展示和模型处理的功能，能让用户在系统里进行模型分层切片，打印轨迹规划，打印过程动画模拟以及记录模型处理日志等操作。本系统作为增材制造预处理过程云服务化的探索，还要关注到用户的数据管理问题，用户通过注册登录系统，建立个人在线模型仓库，便于将模型数据存放在云服务器上，随时修改模型相关信息，也可共享给其他用户或下载到本地使用。此外还需要系统管理员来管理所有用户和模型的信息。详细功能需求说明如表 3-1。

表 3-1 系统功能性需求

需求名称	需求说明
登录	用户输入账号密码登录进入系统，并且对非法数据进行校验
注册	用户输入相关参数注册账号，并对参数进行校验
模型列表	用户在模型列表里可以查看到模型相关信息
上传模型	输入模型相关信息并将相关数据存入数据库
修改模型	修改模型相关信息，并更新数据库对应集合
删除模型	删除模型数据，并更新数据库对应集合
加入仓库	将公共模型数据加入我的模型库内，更新对应数据库信息
个人中心	可以查看到用户个人相关信息以及对信息进行修改
模型文件导入导出	向服务器获取模型数据并导入应用或者将应用数据导出本地
图层控制	将三维场景中的对象进行分类，不同种类可以控制显示隐藏
场景操作	可以对三维场景进行还原、放大、缩小视窗操作
分层切片	对模型数据进行规定参数的分层切片处理
轨迹生成	根据分层切片结果产生的轮廓进行轨迹填充
G 代码	将轨迹数据转化为 G 代码，并可下载到本地
动画演示	将轨迹数据变为可视化动画进行执行
日志记录	将数据处理中每个步骤的输出信息以及执行时间进行记录
用户管理	管理员对系统用户进行查看、修改、删除
模型管理	管理员对系统中的模型进行查看、修改、删除

3.2.2 非功能性需求

非功能性需求指的是全栈开发应用中可以公共抽象化的函数调用。本文使用

Three.js 技术来渲染绘制 3D 模型并构建画面场景，利用 Three.js 提供的 API 来对算法处理后的图形数据进行绘制，同时完成模型的缩放、平移、旋转等交互操作；对于涉及到的算法，本文使用 JavaScript 内置的数据类型进行复现；前端页面采用 Vue 框架进行开发；页面路由控制使用 Vue-Router 进行配置；前端页面之间的数据共享使用 Vuex 进行管理；前后端通信的 HTTP 请求使用 Axios 进行处理；后端使用 Express.js 响应接口并同步操作数据库；数据则使用 MongoDB 数据库进行存储。详细非功能性需求如表 3-2。

表 3-2 系统非功能性需求

需求名称	需求说明
Vue 结构设计	对 HTML、CSS、JS 文件进行关联以及配置代码打包信息
Vuex 数据管理	选择合理的数据结构对页面数据进行双向绑定
接口设计	针对前后端数据通信进行合理封装与错误处理
模型操作抽象封装	对模型对象的获取，显示隐藏，复制数据等公共操作进行封装
数据公有操作封装	算法中需要对数据进行深复制或者闭包暴露等操作进行封装
路由设计	使用 Vue-Router 将页面与访问 Url 合理绑定
JWT 认证	对系统中权限 API 访问资源进行 Token 验证
数据库操作	合理封装数据库的增、删、改、查操作
后端服务	针对每个请求接口提供 GET 或者 POST 方法进行处理封装
性能	该系统要求同时 1000 个并发访问且时延不应超过 2 秒钟
跨平台性	可以在不同的主流浏览器中进行使用

3.2.3 系统用户及用例

用例图（Use Case Diagram）描述了用户如何去使用这个系统，是软件需求分析中十分重要的步骤。基于 Web 的增材制造预处理平台主要分为模型预处理平台和后台数据管理两个子系统，系统用户主要分为普通用户和管理员两类。普通用户在系统里可以完成对模型数据的预处理流程，管理员则是负责维护整个系统的用户和模型信息。下面对系统的用户进行用例分析，并对每个用例进行详细说明。

1. 系统管理员的用例分析

系统管理员用例图如图 3-2。

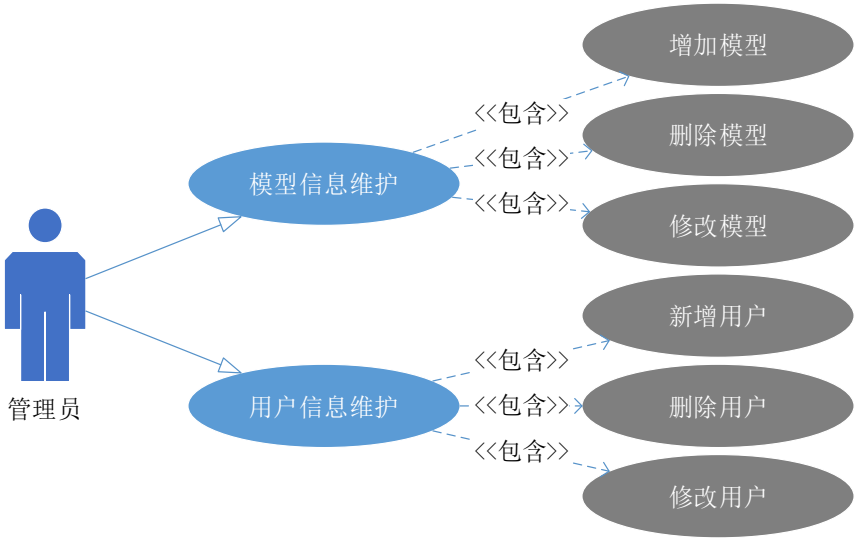


图 3-2 管理员用例图

系统管理员主要负责模型信息和用户信息的维护，模型信息管理包含模型信息的增加、删除以及修改操作；用户信息管理包含对用户基本信息的增加、删除和修改操作。

2. 普通用户的用例分析

普通用户用例图如图 3-3。

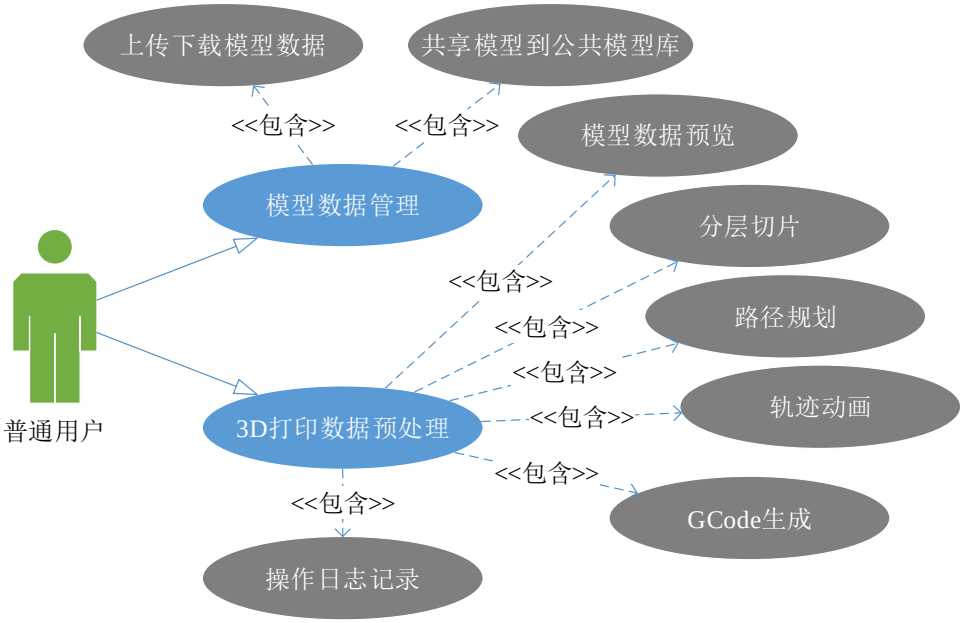


图 3-3 普通用户用例图

针对普通用户的用例进行分析：①用户将本地的模型数据上传到个人仓库中，可

以修改模型相关信息，同时能将模型共享到公共仓库中；②用户在系统里预览模型数据，并提供放缩、旋转、还原、显示和隐藏图层等交互操作。③用户可以使用模型数据的切片分层、轨迹规划、模拟动画和 GCode 生成等功能。④用户在日志列表里查看处理流程的追踪数据，并可将日志记录导出。

3.3 系统架构

根据上文的需求分析可得系统的整体架构：展示层包含 Vue、Ant Design Vue、阿里图标库以及基于 WebGL 的 Three.js；网络层基于 HTTP 协议的 Ajax 进行前后端数据通信；业务层包含用户管理、模型数据管理、数据预处理、模型数据展示等；数据层包含存储过程、数据缓存、自定义函数、读写数据库、事务等；数据库则使用 MongoDB 进行存储；运行环境层包含运行平台环境 Node.js 和物理环境云服务器。详细系统架构如图 3-4。

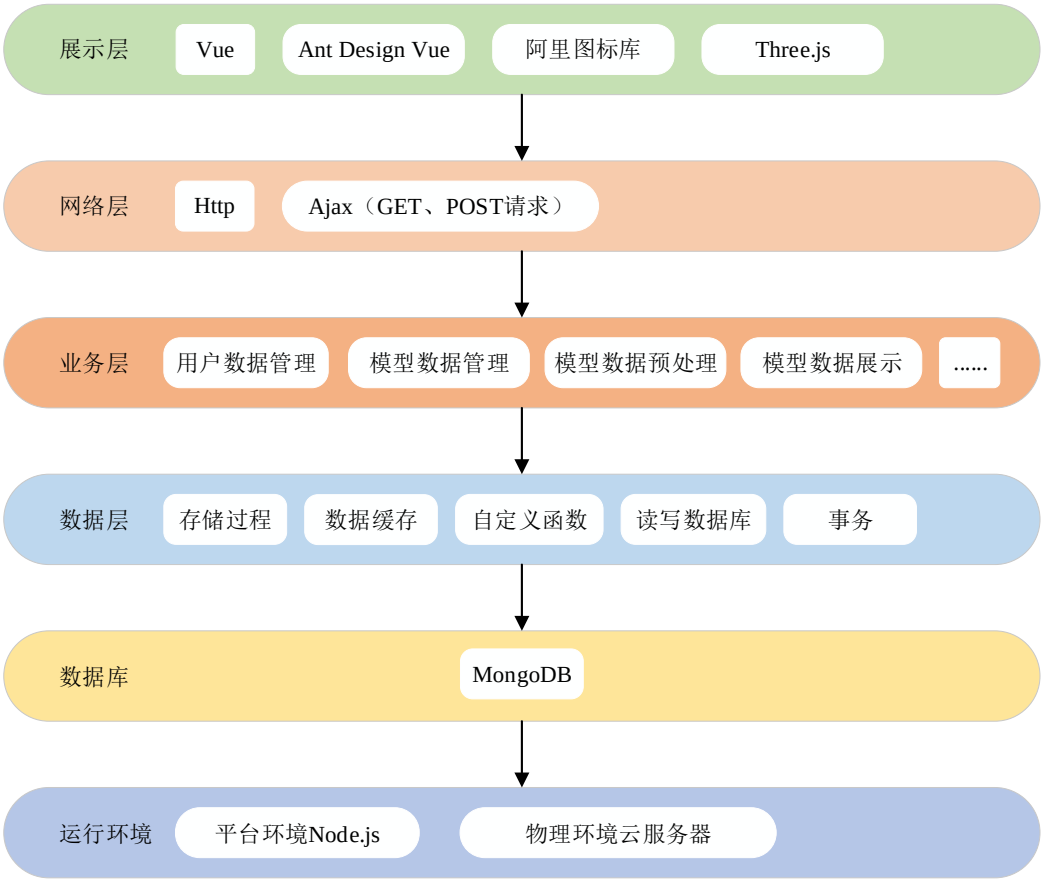


图 3-4 系统架构图

3.4 系统功能模块设计

3.4.1 注册登录模块

注册模块具体流程为：注册页面输入用户名、真实姓名、密码三个数据并点击提交按钮，将数据交付于服务器，服务器先对用户名进行查询，如果有重名用户则返回注册失败，否则将数据插入到用户集合当中，并返回注册成功，具体流程如图 3-5。

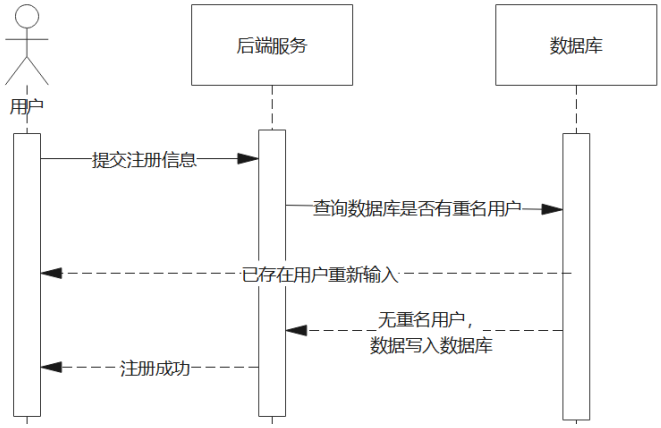


图 3-5 注册时序图

登录模块使用了 JWT（Json web token）进行校验，具体流程为：每次登录时服务器会根据规范生成一个令牌（Token），本文选择的令牌生成算法是 HS256，并把该令牌放在请求头部返回给客户端，此时用户在之后的请求中都把该令牌带上，这样就能保证服务器识别出各个客户端用户的身份。具体验证流程如图 3-6。

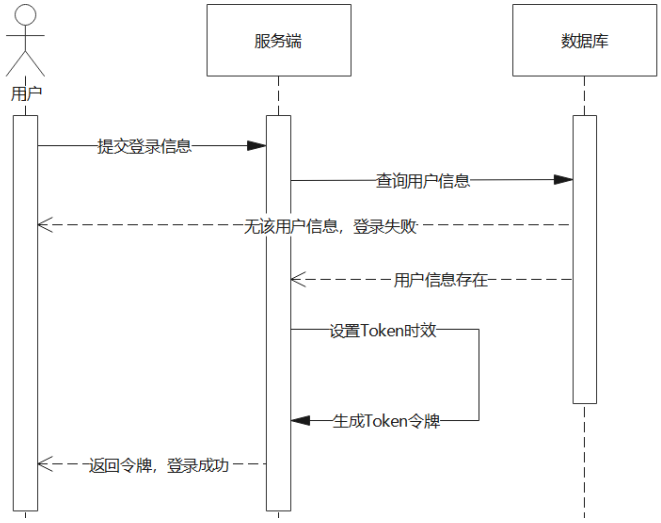


图 3-6 登录验证时序图

3.4.2 用户模型管理模块

用户模型管理模块分为我的模型库和所有模型库，在我的模型库里，用户可以上传本地模型文件到我的模型库，上传时需要输入模型名称、模型描述、模型权限、模型图片、模型文件五个参数，模型上传成功后还能对模型数据进行修改和删除操；在修改模型信息页面中包含模型图片和模型数据的下载功能；所有模型库展示了系统中所有用户上传的模型文件，对于公开权限的模型，用户可以直接将模型加入我的模型库，而私有模型则需要向该用户提出使用申请，如果申请被同意则会自动加入到我的模型库中。模型库管理模块具体功能如图 3-7。

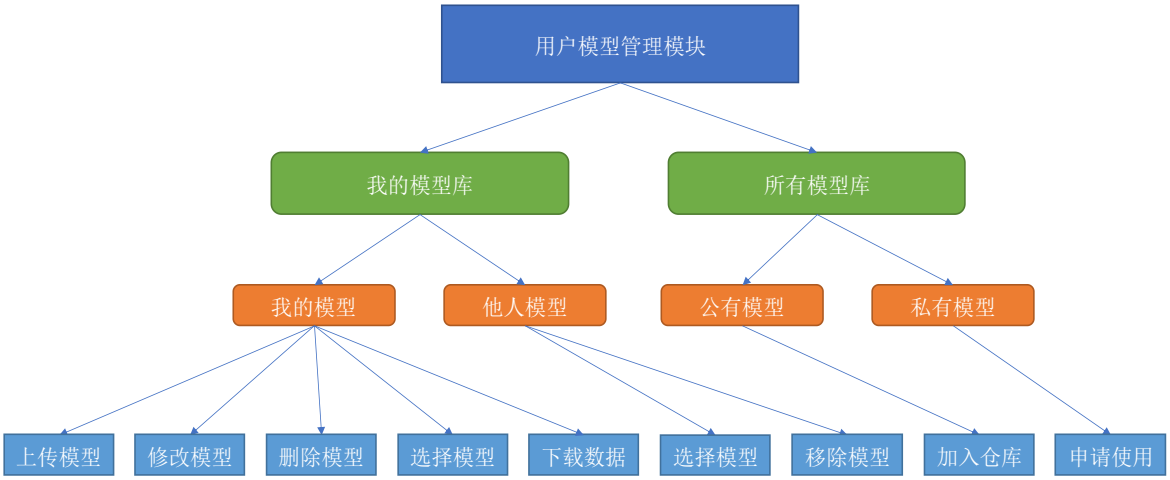


图 3-7 模型管理模块功能示意图

3.4.3 模型显示交互模块

本文使用 Three.js 这个开源库实现模型的展示交互操作，它帮助开发者二次封装了许多 WebGL 底层的 API，避免多次组合封装常规的图形操作，提高开发效率。模型展示与交互部分主要分为基础显示、视窗放缩、视窗回中、图层显示与隐藏、算法处理结果、页面性能面板和动画演示，其中图层的显隐控制包含的对象有包络盒、目标模型、底部网格、坐标轴、水平切片、切片轨迹；而算法处理结果则包含了轮廓线信息以及轮廓填充轨迹信息。具体模块关系如图 3-8。

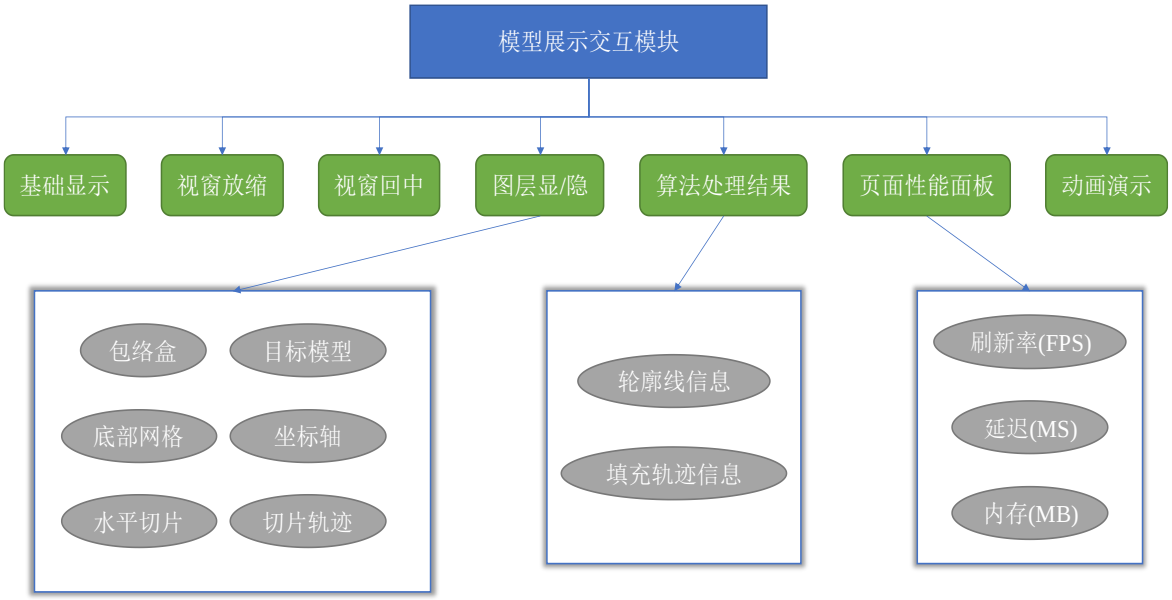


图 3-8 模型展示交互模块示意图

3.4.4 增材制造预处理模块

第二章里分析了增材制造预处理过程所需要的几个步骤，首先导入 STL 模型文件，对导入的模型进行冗余去除和拓扑重建，再使用重建后的模型数据生成切片，然后对每个切片的轮廓进行轨迹填充，最后将轨迹代码转化为打印机可执行的 GCode。增材制造预处理的模块设计如图 3-9。

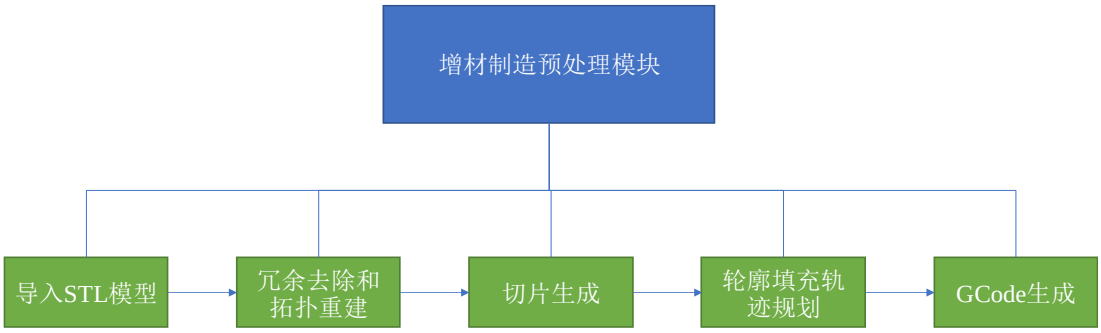


图 3-9 增材制造预处理模块示意图

3.4.5 系统用户信息管理模块

系统用户信息管理模块供管理员使用，包括用户信息搜索，用户信息查看，用户信息修改，用户信息删除。其中搜索功能可以根据用户名与真实姓名对用户数据库进行联合模糊查询，方便管理员对系统用户进行管理，详细结构如图 3-10。

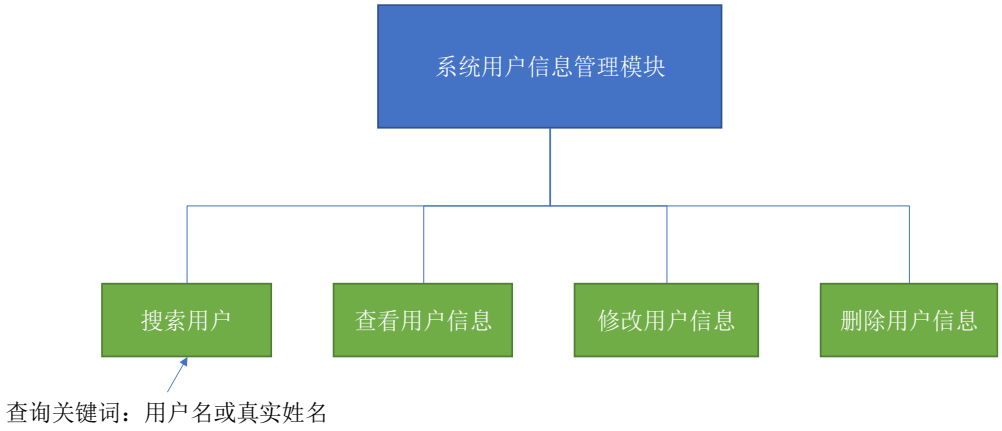


图 3-10 用户信息管理模块示意图

3.4.6 系统模型库管理模块

系统模型库管理模块供管理员管理平台中所有模型数据，包含模型查询，模型数据修改，模型删除功能。其中模型搜索关键词参数有用户名、模型名称或模型描述，具体结构如图 3-11。

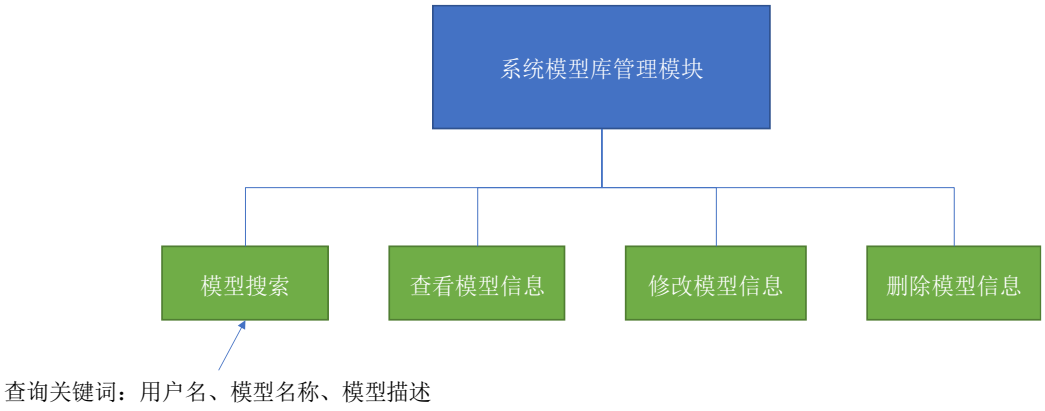


图 3-11 系统模型库管理模块示意图

3.5 本章小结

本章首先介绍了基于 Web 的增材制造预处理平台的整体概述，并对软件进行需求分析，需求分析包含功能性需求、非功能性需求以及系统用例分析，然后根据需求分析设计出系统具体架构，之后对系统进行子功能模块设计，并结合 Web 开发的特点详细设计每个模块的内部流程。

4. 核心算法模块设计

4.1 冗余去除和拓扑重建算法

WebGL 引擎渲染的 STL 模型数据格式是一个个无规律的三角面片，仅存储三点坐标信息，无空间拓扑关系，并且当面片空间涵盖范围远远小于所需精度值时，这些面片可忽略。因此在冗余去除和拓扑重建前，需要对三角面片(Faces)、三角边(Edges)和顶点(Points)三个数据结构进行设计。为了方便后续的切片分层以及路径规划处理，数据结构应满足以下几点特性：

1. 点存储结构记录了点坐标信息以及点索引；
2. 边存储结构记录了边索引、构成边的点索引和每条边的邻接面索引信息；
3. 面存储结构包含了组成面的点索引，组成面的边索引以及面本身的访问索引。

具体数据结构定义如图 4-1。

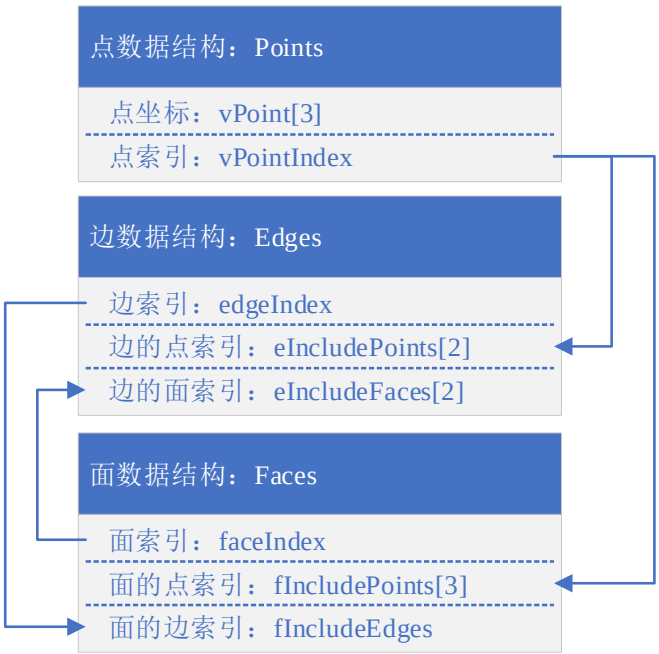


图 4-1 拓扑重建数据结构

因为需要对这些数据进行频繁的查询操作，所以使用散列表的 Map 结构（也就是键值对集合）来存放重构后的数据，其中键值对构建映射关系的方法是哈希函数。

可以把这个 Map 结构看作一个哈希表，哈希函数的选取会直接影响到程序的访问效率。在 STL 文件中，顶点数据通常是 Float 类型。本文选择的哈希函数如式 4-1。

$$\left[|p.x|*\frac{10^4}{21}+|p.y|*\frac{10^4}{19}+|p.z|*\frac{10^4}{17}\right]\bmod(L) \qquad (式\ 4\text{-}1)$$

其中p.x, p.y, p.z为顶点在笛卡尔坐标系中 x, y, z 方向的坐标分量，L为哈希表的长度。

冗余去除是对原始三角面片数组进行循环遍历。在遍历过程中，对每个三角面片顶点进行读取，每读入一个顶点，首先判断该顶点是否在顶点数据结构中已经存在。如果已经存在，则存储该顶点的索引值到三角面片的数据结构中，否则表示该点为新点，存储进顶点表中。在判断顶点的过程中，会发现有些三角面片中点的距离其实很小。这里规定如果两个点的线段距离小于某个设定值Min（本文阈值为 10^{-5}），则可将这两个点当作同一个点来处理。其数学判别式如式 4-2。

$$\sqrt{(p1.x-p2.x)^2+(p1.y-p2.y)^2+(p1.z-p2.z)^2}\le Min \qquad (式\ 4\text{-}2)$$

其中p1、p2为两个顶点，p1.x, p1.y, p1.z和p2.x, p2.y, p2.z分别为p1、p2点在笛卡尔坐标系中的 x,y,z 坐标。

拓扑重建算法流程如下：

1. 声明点、边、面数据结构并初始化；
2. 按顺序读入三角面片信息；
3. 对三角面片的三个顶点和三边计算哈希值，边的哈希值用临时变量存储起来，点数据则存入点哈希 Map 中；
4. 从临时存储的三边信息中取出一组计算得出每条边的长度，如果小于设定的阈值，则继续步骤 4，大于设定的阈值则将边信息存入边哈希 Map 中，如果边信息读取完毕则进入步骤 5，反之继续执行 4；
5. 如果三边信息都存入了边信息 Map 中，则将三个顶点的索引和三边索引存入面片数组的点索引与边索引中，并且把当前面片信息存储到边数据结构中的面索引中；
6. 判断三角面片是否读完。未读完则返回步骤 3；
7. 完成拓扑重建。

在 Web 端实际处理过程为：首先提前定义三个变量 resPoints（Map 结构）、resEdges（Map 结构）和 resFaces（数组结构）分别存储点数据、边数据和面数据，再定义两个哈希生成函数 pointHash 和 edgeHash 分别对点和边进行 Hash 索引值的生成，然后开始遍历模型数据，每个面片数据获取完毕后计算三个点之间的距离，如果小于设定的阈值 T（本文阈值为 10^{-5}），则将两个点合并为一个点（例如 A 点与 B 点，只将 A 点存入 resPoints 内），并把点数组内所有 B 点用 A 点替换，以此类推处理面片的三个点，如果三个点都大于距离阈值，则判断三个点的 Z 轴最大和最小高度差是否也大于阈值 T，如果大于，就将三边与当前面数据存入 resEdges 和 resFaces 中，在存储 resEdges 时除了存储每条边的两个顶点，还应存储边的两个面索引，方便后续寻找邻接三角面片；不满足则处理下一个面片，直至所有面片被处理完毕。具体处理流程如图 4-2。

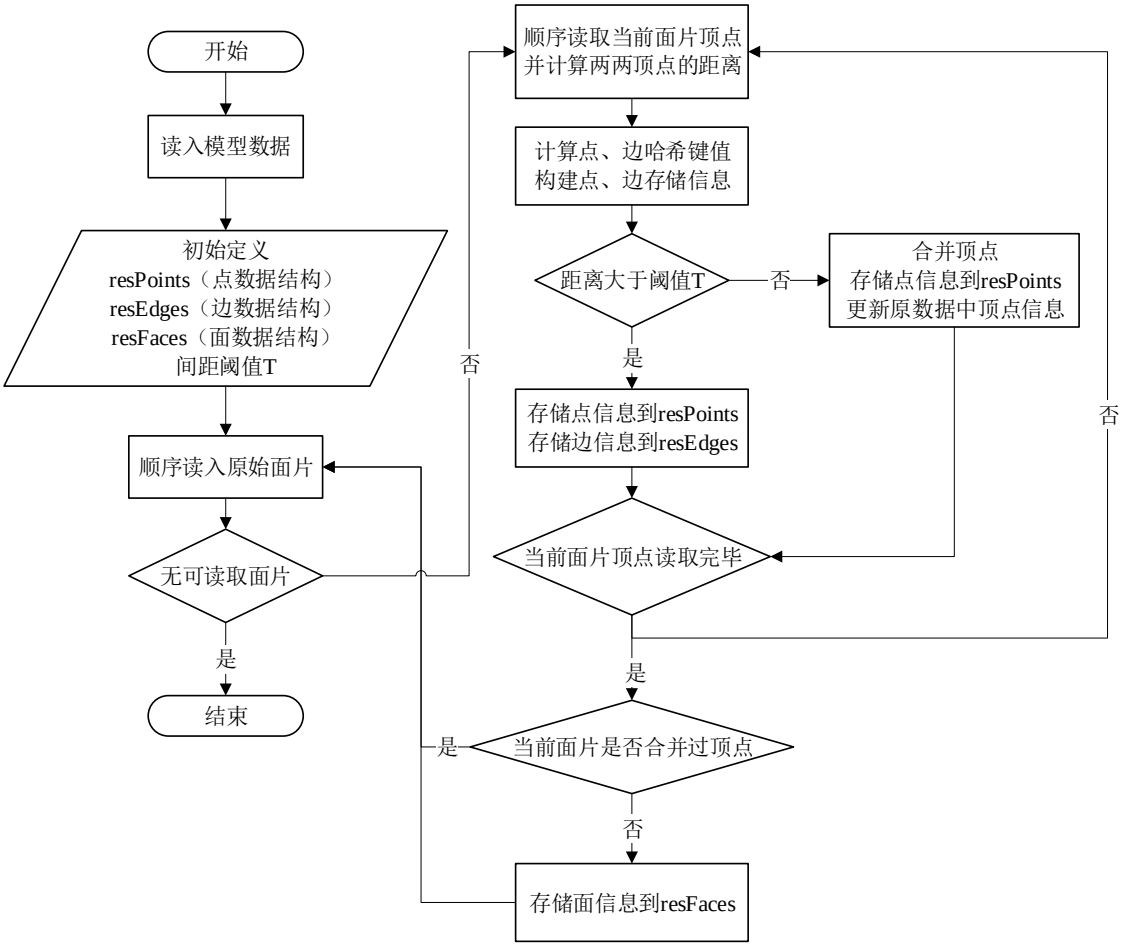


图 4-2 冗余去除与拓扑重建处理流程

冗余去除核心代码为：

```
if(distancePoints(pointA, pointB) > T) {
   resPoints.set(pHashA, {
      vPoint: pointA, //  点坐标
      vPointIndex: pHashA, //  点索引
   });
   resPoints.set(pHashB, {
      vPoint: pointB, //  点坐标
      vPointIndex: pHashB, //  点索引
   });
   resEdge.set(eHashAB, {
      includePoints: [pHashA, pHashB], //  包含的点索引
      includeEdge: eHashAB, //  边索引
   });
}else {
   resPoints.set(pHashA, {
      vPoint: pointA, //  点坐标
      vPointIndex: pHashA, //  点索引
   }); //  只存储第一个顶点，表示两点合并进点哈希存储结构中
   updateOriginData(pointA, pointB); //  更新原始模型数据
}
```

拓扑重建核心代码为：

```
let currentFaceIndex = resFaces.length - 1; //  当前面片索引
let edgeABHash = resEdge.get(eHashAB); //  根据边哈希值获取对应边数据信息
let edgeBCHash = resEdge.get(eHashBC);
let edgeCAHash = resEdge.get(eHashCA);
//  根据边哈希值返回每条边相邻的两个面片，构建边与面的空间拓扑关系
const returnNewFaces = (edgeABHash) => {
   let res = [currentFaceIndex];
```

```
    if (edgeABHash) {
      res = edgeABHash.includeFaces;
      res.push(currentFaceIndex);
    }
    return res;
}
// 更新每条边的邻接面信息，记录拓扑结构信息
resEdge.set(eHashAB, {
    includePoints: [pHashA, pHashB], // 包含的点索引
    includeEdge: eHashAB, // 边索引
    includeFaces: returnNewFaces(edgeABHash), // 每条边包含相邻两个面片
});
resEdge.set(eHashBC, {
    includePoints: [pHashB, pHashC], // 包含的点索引
    includeEdge: eHashBC, // 边索引
    includeFaces: returnNewFaces(edgeBCHash), // 每条边包含相邻两个面片
});
resEdge.set(eHashCA, {
    includePoints: [pHashC, pHashA], // 包含的点索引
    includeEdge: eHashCA, // 边索引
    includeFaces: returnNewFaces(edgeCAHash), // 每条边包含相邻两个面片
});
```

4.2 分层切片算法

本文采用基于拓扑结构的等厚分层切片算法，切片前需要输入分层方向、起始分层高度、终止分层高度和层厚四个参数。从起始高度进行切片时，先判断高度是否与模型有交点，如果有交点则进行邻接三角面片搜索，得到当前高度下的切片轮廓，再进行下一个高度的搜索直到搜索区间内的高度全部被搜索完毕。基于拓扑结构的等厚分层切片算法的处理流程如图 4-3。

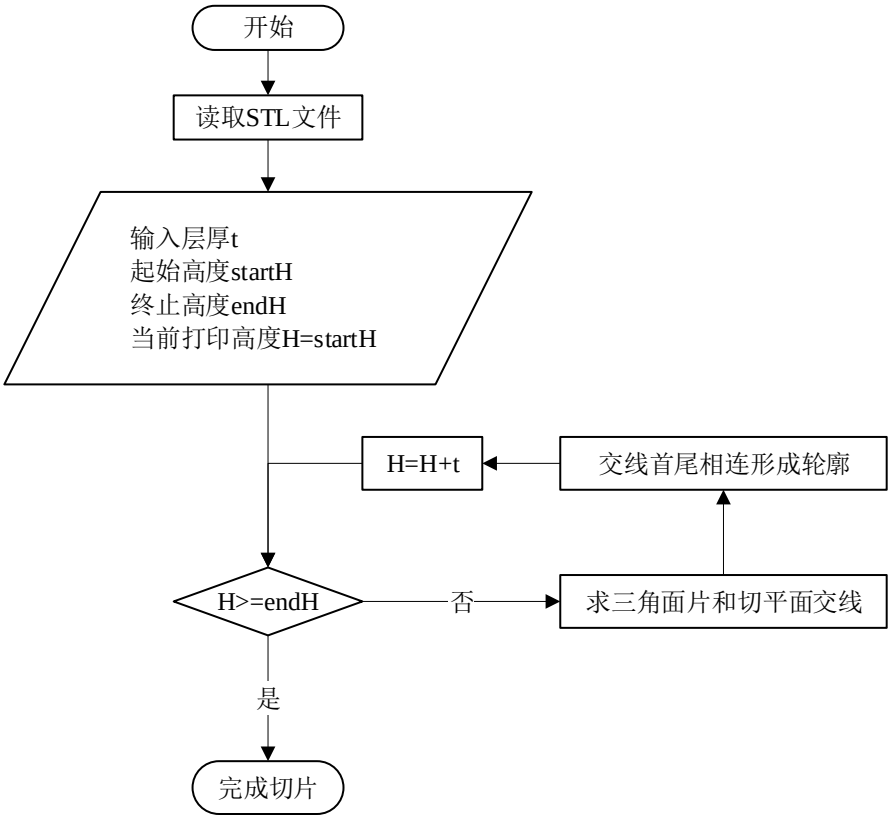


图 4-3 基于拓扑结构的等厚分层切片算法

在上述算法流程中，求出三角面片和切平面交线后，通过边结构中的邻接面去找到下一个与切平面相交的点，从而形成切平面与三角面片的交线，以此类推，就可以得到当前高度的封闭二维轮廓线数据，具体算法流程如下：

1. 声明一个轮廓线存储数组 result，并将提前拓扑构建完成的面数组中的每个数据打上 hasSearch 的标记，并初始化为 FALSE，表示还没有被遍历过；

2. 从初始高度 H 开始，任意寻找一个未被搜索过的且包含当前切片高度范围的三角面片，如果有则进入步骤 3，并记录起始面片索引 startIndex，没有则进入步骤 7；

3. 对该面片的三条边求交点，如果边与切平面共线则跳过该边，继续对其他边进行求交点；

4. 如果得到一个交点则将该三角面片的 hasSearch 设置为 TRUE，如果有两个不同的交点，则将其存放进一个临时数组 temp[2]中，数组分别是两个点的信息，同时将数组存进 result 数组中，然后将该三角面片的 hasSearch 设置为 TRUE；

5. 将步骤 4 中 temp 数组的最后一个点所在的边中的邻接面找到，并进入步骤 3，直至该邻接面的索引为 startIndex；

6. 将高度 H 加上层厚 t，继续步骤 2，直到 H 大于终止高度；

7. 完成对文件的切片。

由于本系统不仅实现分层切片算法还要考虑对轮廓线的绘制以及每层数据的可视化，因此需要针对可视化部分进行改进，下面是算法具体实现：

1. 首先输入起始切片高度、终止切片高度、层厚、以及切片绘制颜色；

2. 进入切片函数后先清除历史切片的数据缓存，再计算分层的背景图层。背景图层为一个范围正好是模型在 XY 轴平面投影 1.5 倍的半透明矩形，生成切片矩形的方法为：先通过初始切片高度和终止切片高度算出切片层数，循环生成 PlaneGeometry 对象，设置每个矩形位置再将每个 PlaneGeometry 实例化为 Mesh 对象存放进一个 Group 中，最后将整个 Group 存入当前 Scene 对象中；

3. 切片背景图层生成后，再计算切片轮廓，使用的方法是对高度进行遍历，从 resFaces 中任意取一个与当前高度相交且未被搜索过的面片，然后初始化轮廓点存储数组 resultPoints，存储第一个面片的索引 indexFirst，再对面片中的每条边遍历计算交点；交点分为两种情况，如果边的一个顶点就是交点，则直接将顶点存入 resultPoints 中，否则计算出交点再存入 resultPoints 中；

4. 交点计算完成后将该面打上已经搜寻过的标记，再寻找该边的非当前邻接面，继续对下一个面进行边遍历，直到下一个面的索引等于 indexFirst，结束当前封闭轮廓的计算；

5. 子轮廓计算完成后继续在 resFaces 中任意取一个符合高度且未被搜索过的面片执行上述处理，直到没有符合条件的面片可以取出，结束所有轮廓搜寻。

分层切片实际流程如图 4-4。

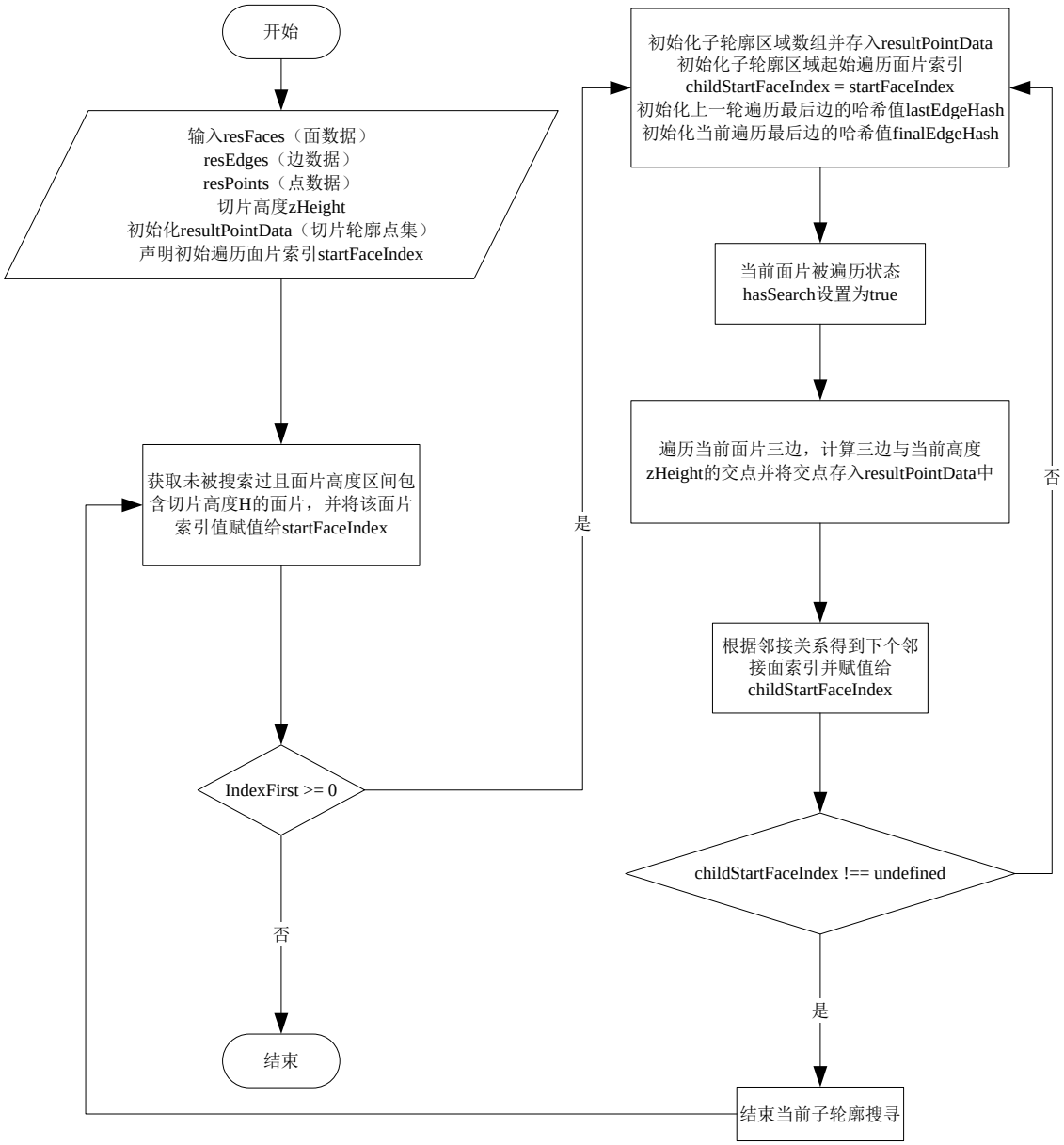


图 4-4 分层切片实际处理流程

分层切片核心代码为：

```
do {
let finalEdgeHash = null; // 初始化当前遍历最后边的哈希值
resFaces[childStartFaceIndex].hasSearch = true; // 将当前面片被遍历状态置为 true
// 遍历当前面片三边，计算三边与当前高度 zHeight 的空间关系
resFaces[childStartFaceIndex].includeEdge.map(edgeHash => {
    let pointA = resPoints.get(resEdge.get(edgeHash).includePoints[0]).vPoint;
```

```
let pointB = resPoints.get(resEdge.get(edgeHash).includePoints[1]).vPoint;
const p1z = Math.formatFloat(pointA.z, 5); // 将 z 轴坐标进行精度近似处理
const p2z = Math.formatFloat(pointB.z, 5); // 将 z 轴坐标进行精度近似处理
if (p1z === zHeight && p2z !== zHeight) { // zHeight 切平面经过边的左端点
 if (lastEdgeHash === null || lastEdgeHash !== edgeHash) {
    finalEdgeHash = edgeHash; // 将当前处理边哈希值赋值给 finalEdgeHash
    // 将交点存入当前子轮廓点集数组中
    resultPointData[currentResIndex].push(pointA); }
     } else if (p1z !== zHeight && p2z === zHeight) { // 切平面经过边的右端点
        if (lastEdgeHash === null || lastEdgeHash !== edgeHash) {
         finalEdgeHash = edgeHash;
         resultPointData[currentResIndex].push(pointB);}
           // zHeight 切平面经过边内部
          } else if ((p1z > zHeight && p2z < zHeight) || (p1z < zHeight && p2z >
zHeight)) {
    if (lastEdgeHash === null || lastEdgeHash !== edgeHash) {
    finalEdgeHash = edgeHash;
    resultPointData[currentResIndex].push(unitCal(pointA, pointB, zHeight));}}})
    lastEdgeHash = finalEdgeHash; // 将当前最后处理的边赋值给 lastEdgeHash
        // 作为下一轮中的上一轮遍历的最后边哈希值
        // 寻找下一个邻接面索引
        childStartFaceIndex = resEdge.get(finalEdgeHash).includeFaces.filter(item
=> !resFaces[item].hasSearch)[0];
      // 未找到满足条件的面片则结束当前子轮廓计算
      } while (childStartFaceIndex !== undefined)
```

获得每个轮廓的数据后还需要对轮廓进行可视化，绘制流程为：

1. 声明一个空几何体对象，将轮廓坐标点添加到对象当中；
2. 设置对象渲染材质，里面包含颜色和大小参数；
3. 将对象添加进 Points 点模型对象中，最后把点模型加入最大的 Scene 场景中

执行渲染；

4. 点绘制完毕后还需将点连接成线，点数据转化为 Vectors 对象存入 listPoints 数组里，通过 BufferGeometry 将 listPoints 转化为缓冲类型几何体，设置渲染材质，将对象和材质放入 Line 线模型中，最后将线模型对象加入 Scene 场景中执行渲染。

分层切片可视化流程如图 4-5。

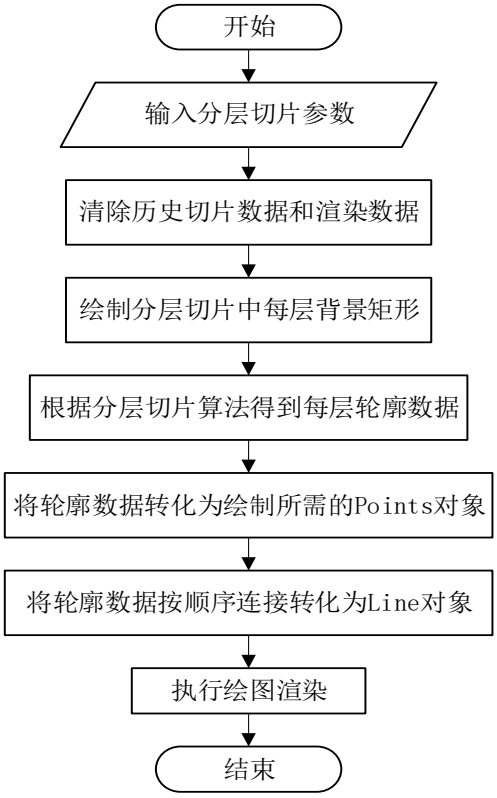


图 4-5 分层切片可视化流程

其中绘制 Points 对象核心代码为：

```
const drawPointByPoints = (data, groupName, color) => {
  const geometry = new Three.Geometry();//声明一个空几何体对象
  data.forEach(item => {
    geometry.vertices.push(new Three.Vector3(item.x, item.y, item.z)); //顶点坐标添加到 geometry 对象
  })
  const material = new Three.PointsMaterial({
    color: color,
```

```
        size: 3
    });//材质对象
    const points = new Three.Points(geometry, material);//点模型对象
    correctOffSet(points, modalOffSet); // 添加偏移修正
    findObjectByName(groupName).add(points); // 将点对象加入特定名称对象中
    render();
}
```

绘制 Line 对象核心代码为：

```
const drawLineByPoints = (data, name, color) => {
    const listPoints = data.map(item => {
        return new Three.Vector3(item.x, item.y, item.z)
    });
    const geometry = new Three.BufferGeometry().setFromPoints(listPoints);
    const material = new Three.MeshBasicMaterial({ // 设置渲染颜色和渲染模式
        color: color,
        side: Three.DoubleSide,
    });
    // Create the final object to add to the scene
    const lineObject = new Three.Line(geometry, material); // 根据 Points 对象构建 Line 对象
    correctOffSet(lineObject, modalOffSet); // 添加偏移修正
    findObjectByName(name).add(lineObject);
    render();
}
```

4.3 轨迹填充算法

本文是基于填充路径的扫描线多边形填充算法，基本原理是用水平线对轮廓进行扫描求交（由上到下），再将交点排序输出，完成填充工作。算法示意如图 4-6。

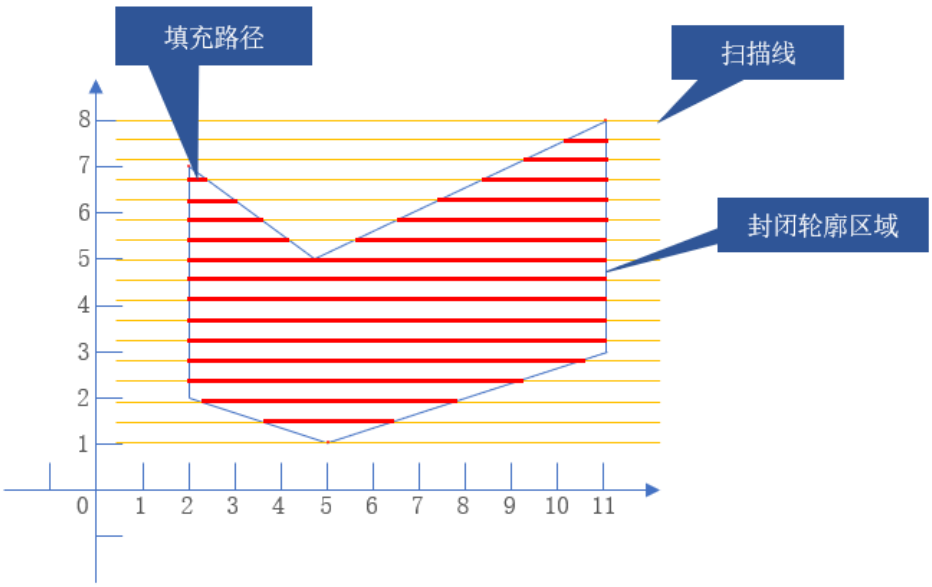


图 4-6 扫描线多边形填充示意图

在轨迹填充执行前需要对轮廓数据格式进行重构，将以点信息为基础的旧轮廓数据转化为以轮廓属性为主体的新轮廓数据。下面是新旧轮廓数据格式定义：

1. 旧轮廓数据存储格式：[轮廓 1:[点 1，点 2，点 3……]]，轮廓 2:[点 1，点 2，点 3……]……].

2. 新轮廓数据存储格式：[轮廓 1:{xMin，xMax，yMin，yMax，轮廓边数据}，轮廓 2:{xMin，xMax，yMin，yMax，轮廓边数据}……].其中 xMin，xMax，yMin，yMax 分别为轮廓在笛卡尔坐标系下 X 轴方向的最小最大值和 Y 轴方向的最小最大值。

接下来进行内外轮廓判别，先声明一个二维矩阵 distinguishMatrix 存储内外轮廓判别标记，然后初始化 contourInfo 数组用来存储新表达形式的轮廓数据，里面包含了 xMin、xMax、yMin、yMax 和 edgeInfo 参数，代表着每个轮廓的范围以及组成轮廓的边数据，然后先对一个子轮廓数据进行遍历，遍历的过程中就能得到每个轮廓的范围极值和组成的边信息，并存入 contourInfo 数组中；经过上面步骤得到了 contourInfo 数组，然后使用双层循环遍历 contourInfo 数组得到内外轮廓判别矩阵 distinguishMatrix，在循环中将外层循环的当前轮廓与内层循环的所有轮廓范围进行范围对比，如果当前轮廓有被其他轮廓包含则将判别矩阵对应位置加 1 并把那个大

的轮廓索引值和 X 轴范围最小值存入判别矩阵，最后得到的 distinguishMatrix 数据就存放着内外轮廓判别依据。接下来要利用 distinguishMatrix 判别矩阵来对轮廓进行合并操作，开始循环判别矩阵，如果轮廓的判别矩阵对应位置的存储数组长度为偶数则为外轮廓，奇数则为内轮廓，如果是外轮廓则无需处理，是内轮廓则需要寻找到包含它的轮廓中最接近它的轮廓，并将该内轮廓的边数据合并到最接近轮廓的边数据中，同时将该内轮廓的数据从 contourInfo 中删除，内外轮廓判别过程如图 4-7。

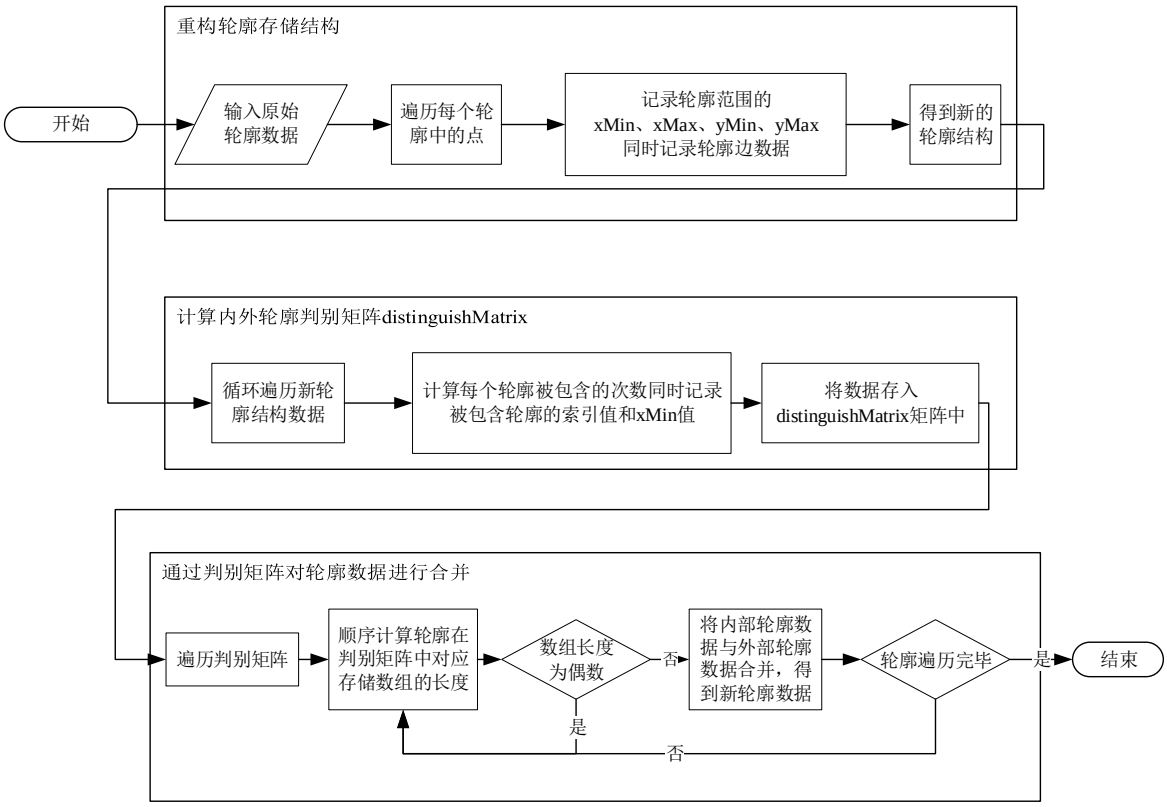


图 4-7 内外轮廓判别过程

得到处理后的轮廓线数据，再通过水平线扫描算法计算出轨迹规划的路径。具体实施流程为：初始化扫描线高度和截止高度为轮廓在 Y 轴的最大值和最小值，再从初始高度向下扫描遍历每个轮廓区域，高度以轨迹密度递增。扫描过程中把每个 Y 轴高度与当前轮廓线求交点，如果交点处于轮廓边之内，则将该交点存入交点数组，如果该交点在两边的最低点则将该交点存入两次交点数组，如果交点位于两边的最高点则跳过该交点，由此得到若干个交点后，按照 X 轴方向从小到大排序，再将每两个点进行连线（例如 ABCDEF 这六个点，得到的轨迹就是 A-B、C-D、E-F 这三条

线段）得到最后的轨迹路径。水平线扫描轨迹填充算法具体流程如图 4-8。

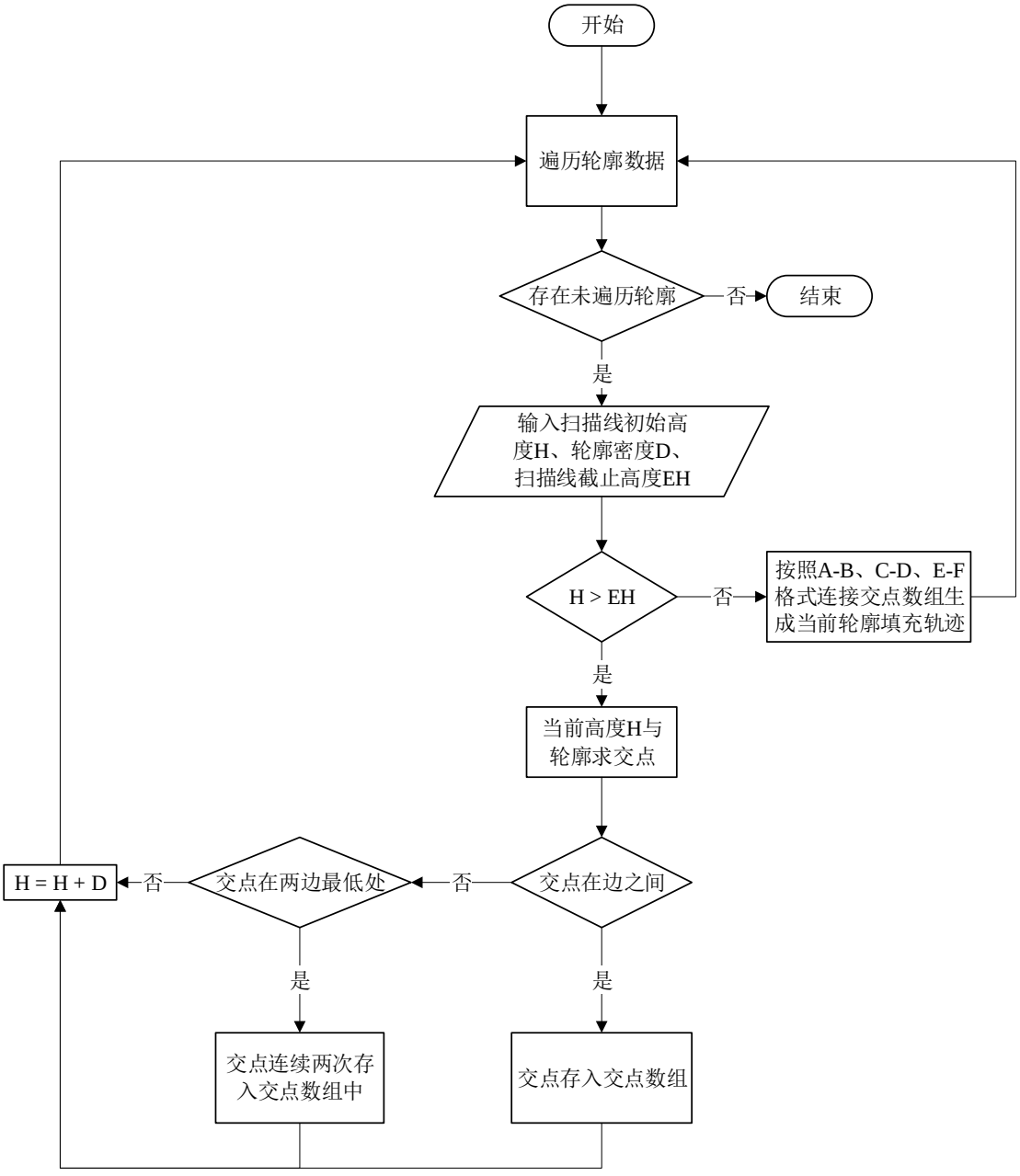


图 4-8 水平扫描轨迹填充流程

水平扫描轨迹填充核心代码为：

```
let startY = contourItem.yMax; //  扫描线初始高度
let endY = contourItem.yMin; //  扫描线截止高度
for (let yHeight = startY; yHeight >= endY; yHeight = yHeight - density) {
```

```
let intersectData = contourItem.edgeInfo.filter(item => item.yMin <= yHeight &&
yHeight <= item.yMax); // 与当前扫描线相交线段数据
let intersectPoints = []; // 交点个数
for (let i = 0; i < intersectData.length;) {
// 交点分为三种情况，在边之内则存入 1 个，在两边构成的凸点处存入 0 个，
在两边构成的凹点处则存入 2 个
if (intersectData[i].yMin < yHeight && yHeight < intersectData[i].yMax) {
intersectPoints.push(unitCalXY(intersectData[i].startPoint, intersectData[i].endPoint,
yHeight)); i++; // 交点在线内直接计算交点并存入 intersectPoints 数组
} else if (i < intersectData.length - 1 && intersectData[i].yMin === intersectData[i +
1].yMin && yHeight === intersectData[i].yMin) {
let intersectionPoint = intersectData[i].startPoint.y - intersectData[i].endPoint.y ?
intersectData[i].endPoint : intersectData[i].startPoint;
intersectPoints.push(intersectionPoint); intersectPoints.push(intersectionPoint);
i += 2;} else {i += 2;}}
intersectPoints.sort((a, b) => a.x - b.x); // 按照 x 坐标排序
// 按照 A-B、C-D、E-F 格式连接
for (let i = 0; i < intersectPoints.length; i += 2) {
   pathPoints.push([intersectPoints[i], intersectPoints[i + 1]])}
}
```

4.4 G 代码生成

3D 打印机使用的 GCode 文件主要由三部分组成，分别为：初始化部分、打印部分、结尾部分。初始化部分设置一些准备参数，比如复位打印喷头和平台的位置，挤出头和打印平台的预热温度，挤出机计数置零等。初始化部分的具体实例化过程为：设置坐标单位、设置喷头和机床温度、喷头和打印平台还原、挤出长度归零、移动喷头到起始位置；打印部分则是对轨迹路径进行转码显示，用来完成实体打印，关注的是打印头的移动速度和喷头送丝量，同时还要根据轨迹的断连控制打印头的升降和送丝的间断。结尾部分是完成打印后关闭打印机的操作指令，比如复位喷头和打印平

台，停止加热材料，关闭驱动电机电源等。G 代码在不同设备和软件下的内容结构都有所不同，因此本文中 G 代码的初始化和结尾部分较为固定，如果有定制化需求，用户可以将 G 代码自行下载修改。

下面分别给出初始化部分、打印部分和结尾部分的简要示例代码。

1. 初始化部分。

G21; 将单位设置为毫米

G90; 采用绝对坐标，所有编入的坐标值全部是以编程零点为基准的

M104 S205; 设置挤出头温度为 205℃

G28; 喷头和底部平台回归原点

G1 Z5 F5000 E1; 沿 Z 轴移动 5mm，速度为 3000mm/min，并挤出 1mm 的丝材

M109 S205; 等待喷头温度到达 205℃

G92 E0; 打印头丝材挤出量归零

M82; 主轴制动器打开

2. 打印部分。

G0 X67.263 Y59.881 Z0.500 F7800.000; 打印头以 7800mm/min 的速度从原点运动到坐标(67.263,59.881,0.500)处，不进行送丝

G1 X68.176 Y59.025 E1 F1080.000; 打印头从上个点以 1080mm/min 的速度移动到坐标(68.176, 59.025,0.500)处，同时匀速送丝 1mm 的长度进行打印

G1 X 68.709 Y 58.616 E2; 打印头从上个点以 1080mm/min 的速度移动到坐标(68.709, 58.616,0.500)处，同时匀速送丝 1mm 的长度进行打印

3. 结尾部分。

G92 E0; 打印头丝材挤出量归零

M104 S0; 停止加热打印头

M140 S0; 停止给平台加热

G28 X0 Y0 Z0; 将打印头回归原点

M84; 电机断电

4.5 动画模拟

动画模拟填充轨迹，目的是对打印流程进行三维仿真，给后续增加复杂的动画演示功能预留接口。利用 Three.js 中的帧动画来实现一个标记球体沿着填充轨迹运动的效果，首先需要定义一个 SphereGeometry 的球体对象，设置球体的大小与材质，再定义一个 CatmullRomCurve3 的 3D 样条曲线，用水平扫描填充计算出来的数据对该曲线进行初始化，然后用 Float32Array 类型数组生成动画时间序列，再利用轨迹路径坐标去生成一个和时间序列相对于的位置坐标序列，得到每个关键帧的时值与当前时值所对应的运动坐标点，再使用 Three.js 提供的 KeyframeTrack 函数生成动画轨迹数据，同时，利用 AnimationClip 和 AnimationMixer 函数绑定球体与路线轨迹，最后调用 play 方法执行动画。实际流程如图 4-9。

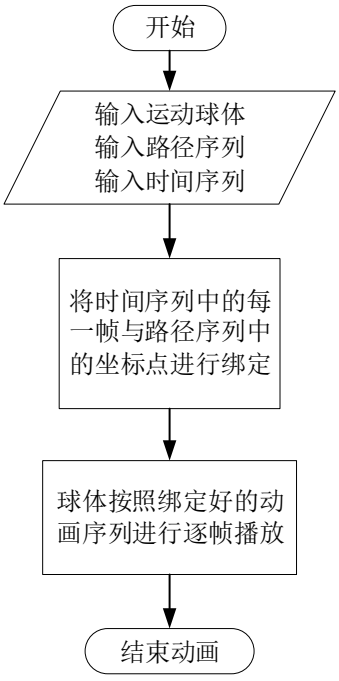


图 4-9 动画模拟轨迹流程

动画绘制核心代码为：

```
// 生成一个时间序列
let times = new Float32Array(arr);
let posArr = []
points.forEach(elem => {
```

```
    posArr.push(elem.x, elem.y, elem.z)
});
// 创建一个和时间序列相对应的位置坐标系列
let values = new Float32Array(posArr);
// 创建一个帧动画的关键帧数据，曲线上的位置序列对应一个时间序列
let posTrack = new Three.KeyframeTrack('.position', times, values);
let clip = new Three.AnimationClip("default", curvePointsLen, [posTrack]);
let mixer = new Three.AnimationMixer(sphere); // 绑定球体与曲线轨迹
let AnimationAction = mixer.clipAction(clip); // 实例化动画操作
AnimationAction.timeScale = 8; // 动画播放速度
AnimationAction.play(); // 执行动画
let clock = new Three.Clock();//声明一个时钟对象
const render = () => {
    renderer.render(scene, camera);
    requestAnimationFrame(render);
    // 更新帧动画的时间
    mixer.update(clock.getDelta());
}
render(); // 页面执行渲染
```

4.6 日志记录埋点

日志可以看作关键操作的数据埋点，记录用户在增材制造预处理过程中所产生的数据信息，为后续结果分析提供参考。因为本系统是基于组件形式的模块化开发，要存储不同组件间执行的结果，需要涉及到组件通信问题。Vuex 库为 Vue 架构代码提供了页面数据共享的解决方案，首先在 store 文件中定义一个 loggingData.js 文件，该文件分为 State、Mutations 和 Actions 三部分，State 用来存放日志总数据，格式定义为{action——操作说明,date——记录时间}；Mutations 内定义同步方法，用来修改 State 中的数据，本文暂时定义了添加新日志记录和重置日志记录这两个函数；Actions

中也是定义一些操作函数来修改 State 的数据，但是函数为异步执行，由于本系统中还没有需要异步记录的数据，因此暂不使用 Actions。触发埋点操作流程如图 4-10。

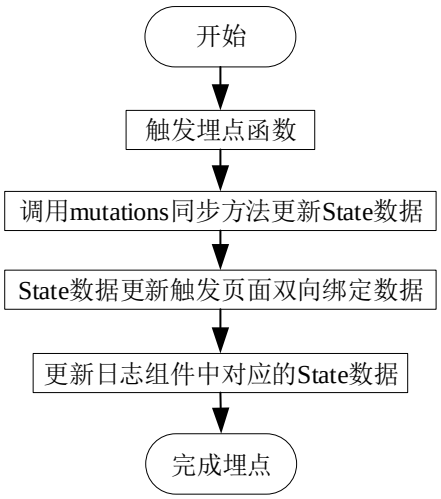


图 4-10 触发埋点操作流程

4.7 图层显隐控制映射树构建

由于 Three.js 中 Scene 的数据结构与右侧显隐控制菜单栏的数据结构不一致，但是在进行显隐操作时两者又要建立关联映射，两者关系对应如图 4-11。其中 Scene 对象数据控制页面三维数据渲染，菜单栏树形数据控制菜单中的选项以及选中状态，它们之间通过 Visible 属性来进行关联映射，当某个数据发生变化，通过显隐控制器去保证双方数据之间的对应关系维持不变。

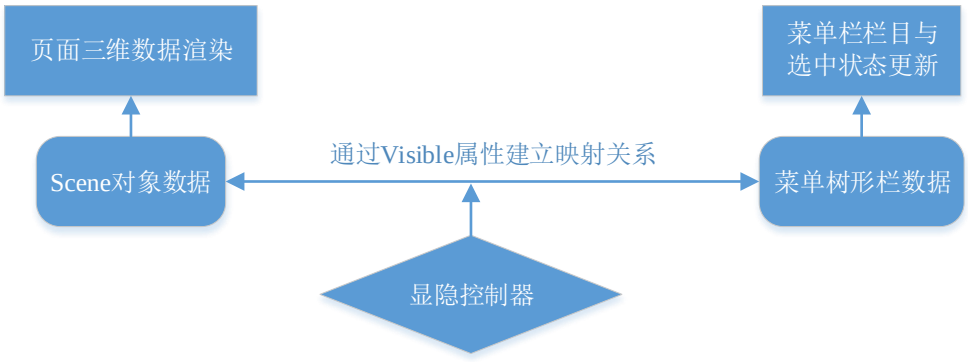


图 4-11 Scene 对象数据与菜单树形栏数据关系

因此场景图层的显隐控制流程分为构建场景对象映射树和点击显隐按钮更新模型显隐状态两部分。构建场景对象映射树的算法实现是：初始化映射树数组和当前显

示图层数组，遍历 Scene 内部的 Children 数组，首先判断数组中的节点是否为 Group 对象，如果是 Group 对象再判断是否为父节点，其次判断是否为可见，只有非父节点且可见的对象才可以加入当前显示的图形数组中，接下来继续判断当前对象是否为叶子节点，是叶子节点则加入映射树数组当中，不是叶子节点则将当前节点的 Children 数组进行递归处理，直至遍历完 Scene 下的每个子 Group 对象，完成映射树和当前显示对象数据的构建。

点击显隐按钮更新模型显隐状态实现思路为：当点击某个图层显示隐藏时，当前显示对象数组会有监听变化，遍历图层所有的叶子节点与当前显示对象数组进行对比，如果叶子节点存在于当前显示对象数组中则执行 showHide 函数将叶子节点的 visible 属性设置为 true，如果不存在则将 visible 属性设为 false，从而实现控制模型图层的显示与隐藏的功能。

4.8 本章小结

本章阐述增材制造预处理在 Web 平台上各个核心算法的设计过程。首先通过第三章的需求分析确定了本系统的几个核心模块，分别为 STL 模型数据冗余去除和拓扑重建、分层切片、轨迹填充、G 代码生成、动画模拟、日志记录埋点、图层控制映射树构建，然后根据 Web 端的实际处理方式与可视化需求设计出较高效的算法，满足在线实时运行需求。

5. 增材制造预处理 Web 平台实现与测试

5.1 开发环境搭建

5.1.1 搭建 Node.js 服务器

由于 JavaScript 的代码需要在 Node.js 的服务端上运行，首先进行 Node.js 的安装，在官网上（https://nodejs.org/zh-cn）选择对应系统的安装包，根据提示完成 Node.js 服务的安装。当前版本的 Node.js 默认包含了 Npm 包管理工具，可以通过命令行输入 node –v 和 npm –v 去测试 Node.js 和 npm 是否安装成功，安装成功后会出现版本的信息。

5.1.2 数据库安装

在 MongoDB 官网找到对应系统安装包，下载完成后配置数据存储位置，数据分为数据库数据和日志数据，本文将数据存放在 C:\MongoDBData 路径下。接下来执行 mongod.exe 文件来安装 MongoDB 服务，在安装路径中的 bin 目录下打开命令行，输入 C:\mongodb\bin\mongod.exe --config "C:\mongodb\mongod.cfg" –install 来安装 MongoDB 服务。最后通过 net start MongoDB 和 net stop MongoDB 来启动和关闭 MongoDB 服务，效果如图 5-1。

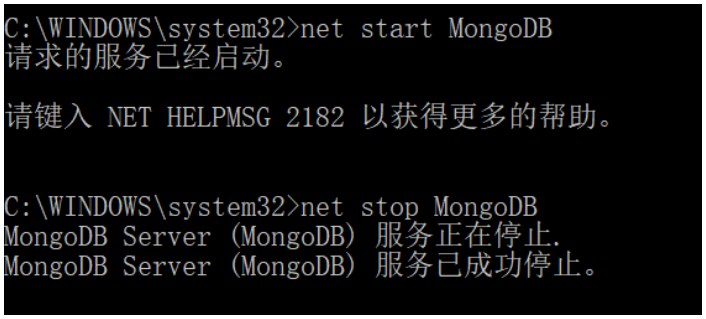


图 5-1 MongoDB 数据库服务启动与停止

5.1.3 软硬件平台

本系统开发使用到的硬件配置如表 5-1，软件配置如表 5-2。

表 5-1 开发硬件配置

序号	名称	配置
1	处理器	Intel(R) Core (TM) i7-7700HQ CPU @ 2.80GHz
2	机带 RAM	8.00GB
3	显卡	NVIDIA GeForce GTX 1050 Ti

表 5-2 开发软件配置

序号	软件	说明
1	Windows 10	操作系统
2	WebStrom	代码编辑器
3	Node.js	JavaScript 运行环境
4	Chrome 浏览器	Web 应用运行环境
5	Three.js	图形开发包
6	Vue	前端开发包
7	Express.js	后台开发包
8	Mongoose	数据库开发包

5.2 前端实现

5.2.1 前端项目初始化

本文前端开发使用到的框架是 Vue，Vue 项目的初始化可以使用官方提供的脚手架工具 Vue-cli，它为开发者提供了一个很好的常规项目模板，避免开发者重复配置打包转码等通用处理操作。首先打开命令行运行官方命令 npm install -g @vue/cli 进行安装，再在项目目录下用命令行执行 vue create web-three 并选择相关配置参数，然后等待项目创建，最后程序执行完毕会看到目录下多出一个 web-three 文件夹，完成项目初始化。进入该文件夹后通过执行 npm run serve 命令即可启动项目。

项目创建成功后使用 WebStrom 打开项目文件，对项目的目录树进行分析。其中 node_modules 文件存放的是整个项目的开发依赖包，public 目录下存放网站图标和入口 Html 文件；src 文件夹下是项目源码，其中 assets 文件夹下存放的是静态资源，而 components 文件夹则是 Vue 的业务组件代码，main.js 是整个项目的入口文件，App.vue

是页面实例化的入口文件；package.json 文件定义了开发所需的各种模块以及项目的配置信息，babel.config.js 文件是 ES6 语法编译配置；最后 README.md 是 Markdown 格式文件，包含项目启动、打包构建等信息。

根据主流的前后端分离架构，前端负责页面展示，业务逻辑控制，数据输入校验，页面路由跳转，页内数据传递以及定义接口字段。这些功能都有不同的开发包来进行处理，为了规范编码流程以及合理管理代码模块，需要改造初始化的项目结构，并引入相关资源依赖包来支撑后续代码的编写，经改造后的项目结构如图 5-2。在 src 目录下新增了 api 文件夹，用来管理应用中的各个 Http 请求；新增 js 文件夹，存放绘图操作于模型处理所涉及到的算法实现代码；新增 routers 文件夹，存放路由控制于每个页面的详细实现代码；新增 store 文件夹，管理页面之间需要传递或者共享的数据；新增 vue.config.js 文件，放置项目构建打包与部署服务器的优化配置代码。重新改造后的目录基本涵盖目前常规 Web 应用开发所能涉及到的模块。

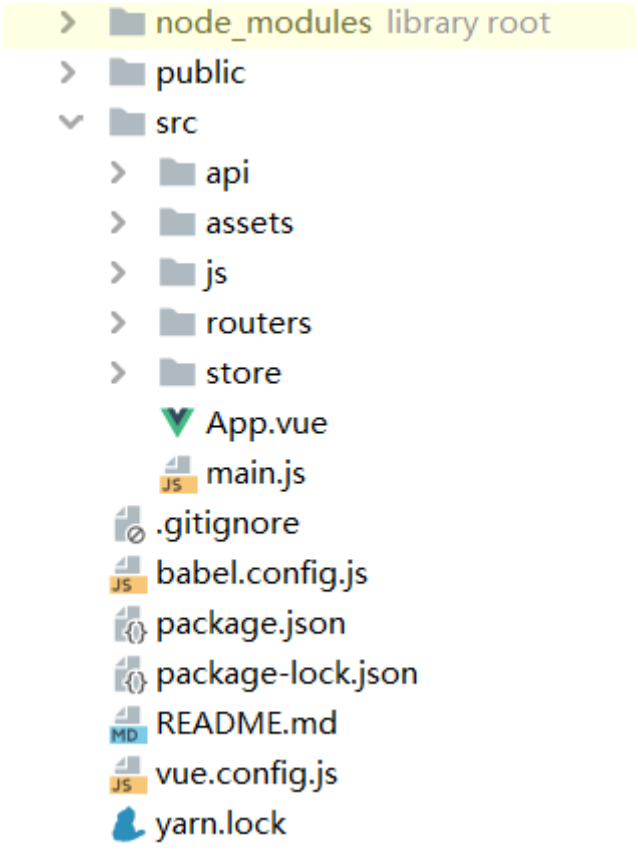


图 5-2 改造后的前端目录结构

接下来进行开发依赖包的选择，在 package.json 中查看本文使用到的公共模块包，一般分为运行依赖包和开发依赖包。运行依赖包如表 5-3，开发依赖包如表 5-4。

表 5-3 运行依赖包

序号	名称	说明
1	ant-design-vue	UI 样式库
2	Axios	基于 Promise 的 HTTP 库
3	core-js	Javascript 的模块化标准库
4	file-saver	文件导出工具
5	Lodash	Javascript 实用数据操作工具库
6	Moment	Javascript 日期处理库，用于解析、检验、操作和显示时间
7	Normalize.css	重置网页 CSS 默认样式
8	Stats.js	WebGL 下的 Javascript 性能监视器
9	Three.js	浏览器 3D 开发库
10	Vue	构建用户界面渐进式框架
11	vue-router	Vue.js 官方路由管理器
12	Vuex	Vue.js 应用程序状态管理器

表 5-4 开发依赖包

序号	名称	说明
1	@vue/cli-plugin-babel	Vue-cli 脚手架转码工具
2	@vue/cli-plugin-eslint	Vue-cli 脚手架代码检查工具
3	@vue/cli-service	Vue-cli 脚手架服务器工具
4	babel-eslint	语法解析器
5	babel-plugin-import	语法解析器
6	Eslint	Javascript 代码检查工具
7	eslint-plugin-vue	Vue 代码检查工具
8	Less	CSS 预处理器，扩展 CSS 语言
9	less-loader	编译 less 文件
10	node-sass	将.scss 文件本地编译为.css 文件
11	sass-loader	编译 sass 文件
12	vue-template-compiler	Vue 的 template 模板编译器

5.2.2 页面路由设计

路由在网页中对应不同 url 下的页面资源，在第三章里介绍了系统的核心流程，本系统的页面主要分为登录页面、模型文件管理页面、工作台页面和管理员页面，因此路由四个模块组成，分别展示对应的页面。在 router 文件夹的 index 文件里进行路由配置，首先设置路由的模式，在 Vue-router 中有两种模式，一种是 hash 模式，一种是 history 模式，两者最大的区别在于 hash 模式的 url 地址是 http://www.mysite.com/#/hello，而 history 模式的 url 地址是 http://www.mysite.com/hello，前者在请求服务器资源时候不会带上#后面的参数，而后者则是请求完整的 url 路径地址，如果后端没有做相应路径的资源配置，就会出现 404 页面。在前后端分离模式下，前端单独运行一个服务器对路由进行处理，因此不会出现刷新页面后丢失资源的问题，且 history 模式更符合 url 地址规范，去除#号后显得更加美观。根据上述页面需求设置以下四个路由：/login 路由下的登录页面、/file 路由下的文件管理页面、/work 路由下的工作台页面和/admin 路由下的管理员页面。

5.2.3 代码编辑器选择

系统实现使用的编程语言为 JavaScript，是一门解释性脚本语言，不需要提前编译，浏览器就能直接运行。为了方便开发者快捷地编码，需要一款具有代码补全、打点调试以及颜色标记等功能的编辑器，通过市场调研选择 WebStrom 作为代码编辑器。这是 JetBrians 公司旗下的一款 JavaScript 开发工具，支持多种语言框架，内部涵盖大部分功能插件，且对许多工具进行了集成，如 npm、git、yeoman 等。软件安装只需在官网（https://www.jetbrains.com/webstorm/）下载对应系统的版本进行安装即可。

5.2.4 HTTP 请求封装

前后端分离模式下，数据传输是通过 RESTful 风格的接口来实现。HTTP 请求方法一共有 9 种，分别是 GET、POST、 HEAD、OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法，本文中用到 GET 和 POST 这两种方法。从 HTTP 层面上来说，客户端收到的状态码是有限的，针对不同的接口都有可能出现这些状态码，因此在调用接口时需要对请求接口做合理封装，并且由于业务因素，相同的接口很可

能会在不同页面多次调用，所以还要根据后端数据类型统一分发调用，让代码更加精简，提高可维护性。

下面来详细介绍 HTTP 请求的封装过程，文件结构如图 5-3：

1. api 文件夹里新建 baseUrl.js 和 axiosService.js 文件，前者配置了不同路由下所控制的资源，后者对所有接口在通信层面上的处理；

2. baseUrl.js 里配置后端服务的 ip 地址，根据后端不同的部署位置进行设置。其中 distributeUrl 对象存放后端服务根地址、用户数据控制地址和模型数据控制地址；

3. axiosService.js 中创建了 axios 实例，首先设置了请求的过期时间和请求头信息，在请求拦截器中将后端返回的 Token 值放入请求头部中，方便后端进行用户权限校验，然后在响应拦截器中处理不同的状态码，如果请求的返回状态码为 200 或者 201 直接将后端返回的数据 resolve；状态码为 401 表示登录过期，将页面跳至登录页面重新登录；状态码为 403 表示接口权限不足；状态码 500 返回服务器内部错误，状态码 502 返回网络问题。

4. apiSugar 里创建 index.js 入口文件分发所有数据控制接口。

5. modelController.js 和 tokensController.js 文件中对每个接口进行封装，以 tokensController.js 文件为例，首先导入接口域名列表和通信层封装后的 axios 接口函数，然后针对不同的请求方法进行 export 处理，其中每个函数接收的 params 形参是需要给后端传递的数据。

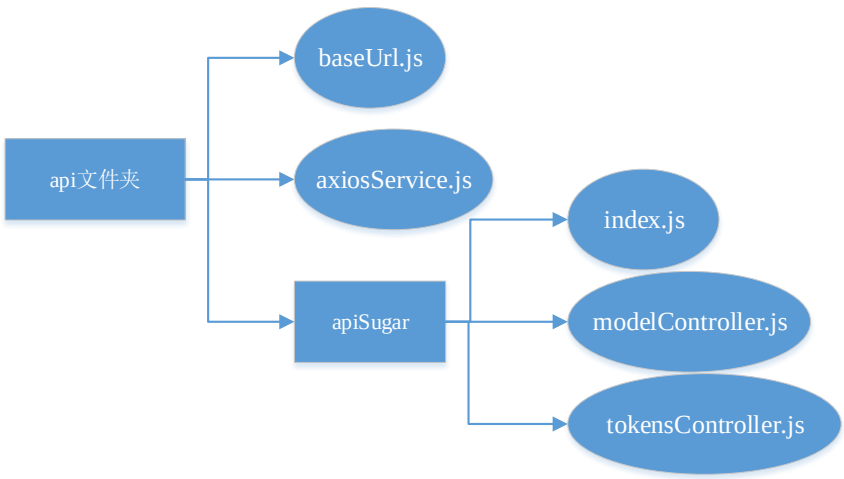


图 5-3 HTTP 请求封装文件结构

5.2.5 系统页面实现

Web 页面都是由 HTML、CSS、JS 三部分组成，在 Vue 框架中通过 Template、Style 和 Script 标签来存放对应代码，其中 Template 标签内编写页面 HTML 骨架代码、Style 标签放置 CSS 代码、Script 标签里存放逻辑控制代码。整个页面样式组件使用蚂蚁金服开源的 UI 框架 Ant Design of Vue，下面是系统各个页面的实现效果：

1. 登录注册页面：包含平台的功能说明，登录表单交互，注册表单交互。

（a）登录页面　　（b）注册页面

图 5-4 登录注册页面

2. 模型管理中心：我的模型库页面，包含模型数据展示、模型数据修改表单、上传模型表单和删除提示；所有模型库页面，包含了模型数据展示、加入仓库和申请使用；个人中心页面，个人资料显示和个人资料修改。

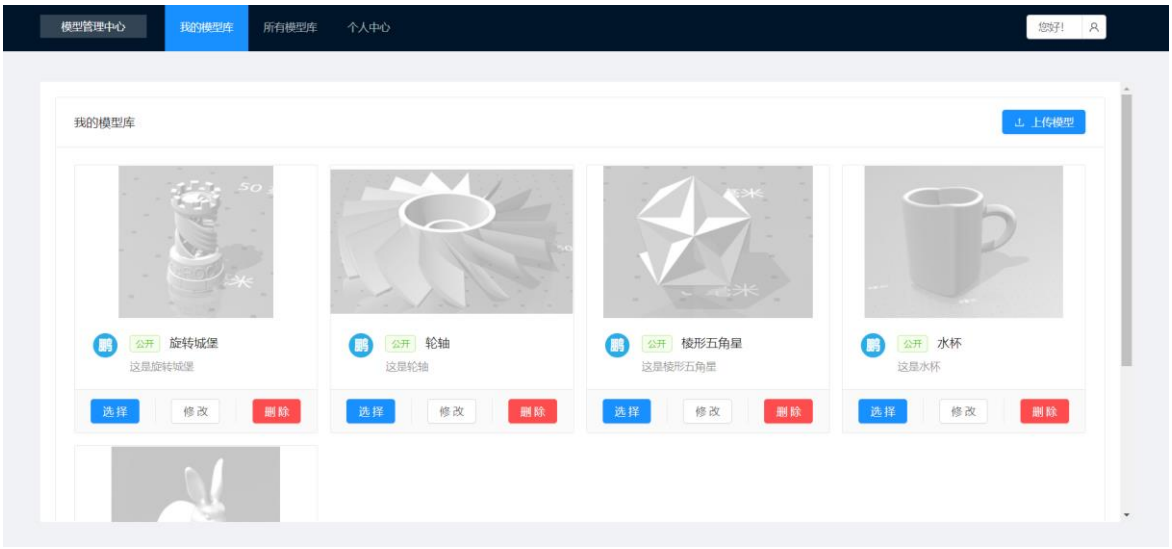


图 5-5 我的模型库页面

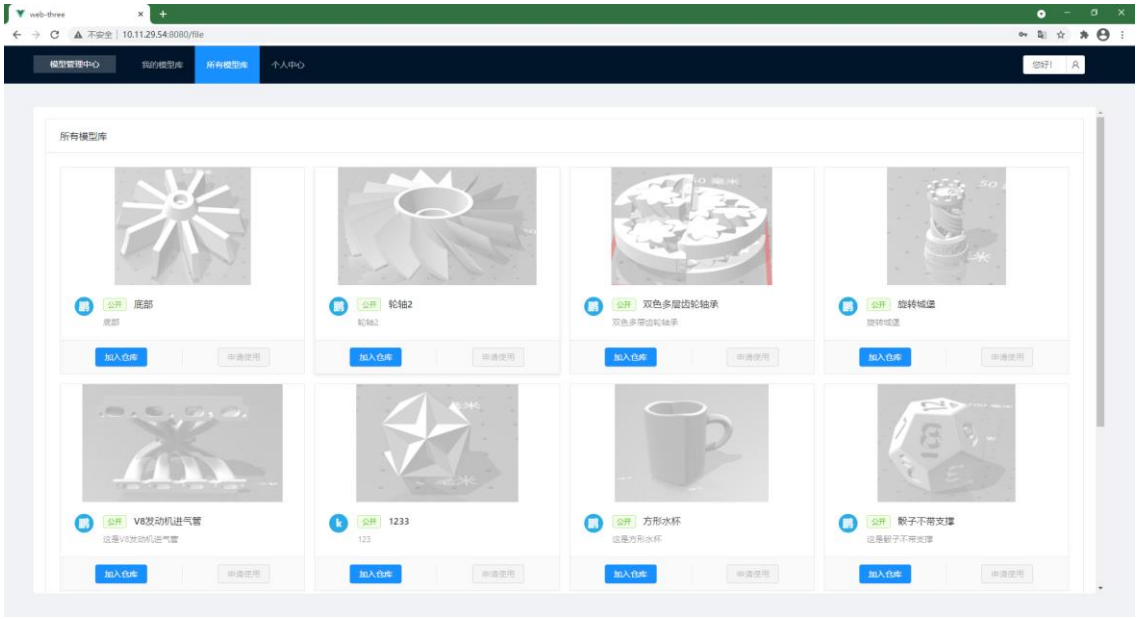


图 5-6 所有模型库页面

图 5-7 个人中心页面

3. 工作台页面：顶部菜单栏（分为文件、图层、功能，文件有另存为、退出编辑，图层有包络盒、目标模型、底部网格、坐标轴等显/隐控制，功能包括切片、轨迹生成、G 代码、动画和日志）；左侧菜单栏（含有还原、放大、缩小操作）；右上方有 JavaScript 性能监听器（包含帧率、延时、内存监听）。

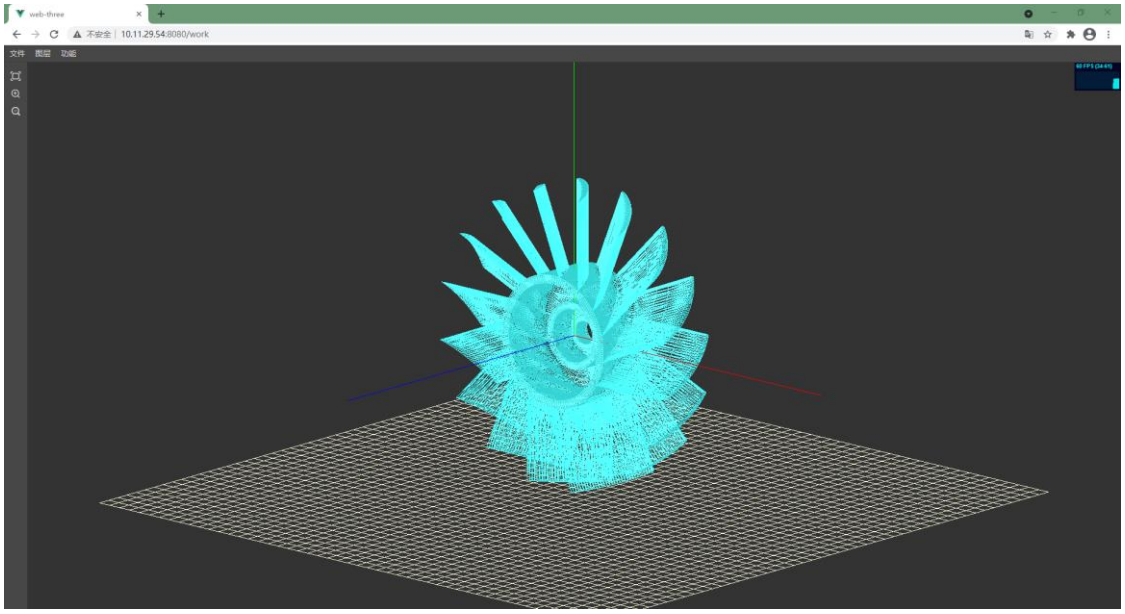
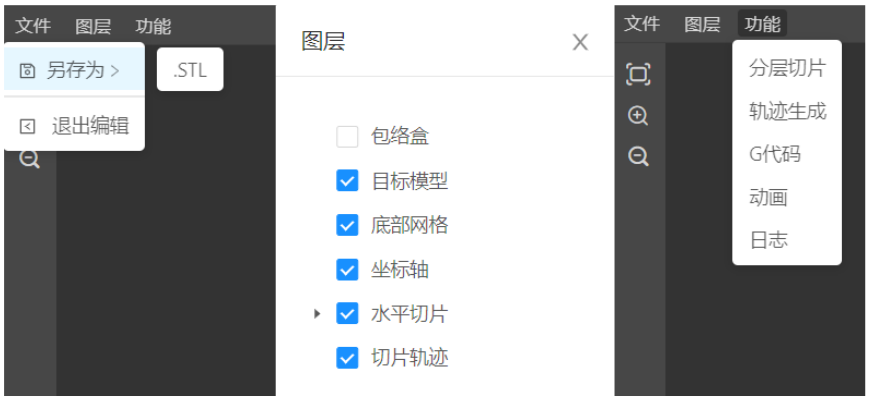
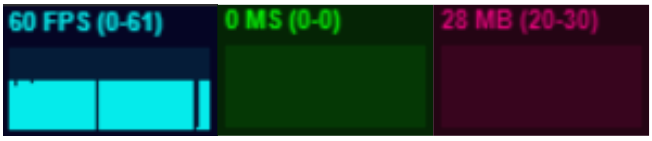


图 5-8 工作台页面



（a）文件菜单　　（b）图层菜单　　（c）功能菜单

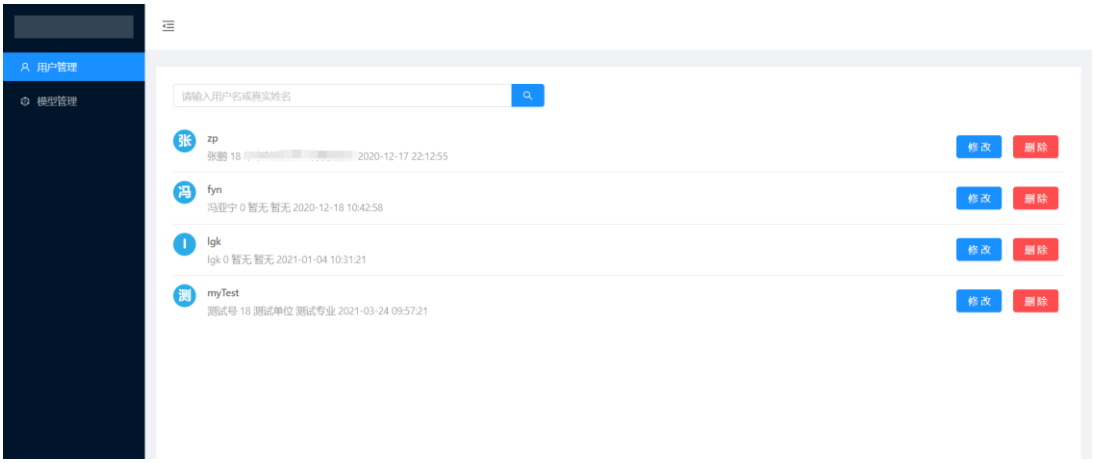
图 5-9 顶部菜单栏



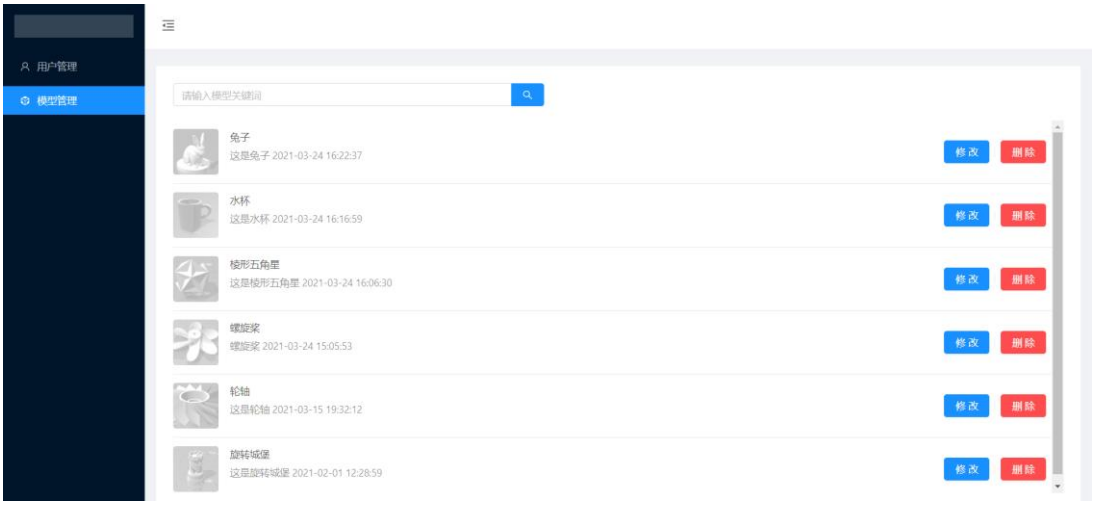
(a)帧率　　（b）时延　　（c）内存

图 5-10 JavaScript 性能监听器

4. 管理员页面：分为用户管理页面和模型管理页面，用户管理页面包含系统用户的增、删、改、查，模型管理页面包含 STL 模型文件的查看和删除功能。



（a）用户管理界面



（b）模型管理界面

图 5-11 管理员页面

5.3 模型交互实现

三维图形点云处理与二维图像元素处理方式差别较大，因此在使用 Three.js 前需要掌握 3D 编程的基本概念，主要包括以下三个要素：场景（Scene）、相机（Camera）和渲染器（Renderer）。Three.js 程序结构如图 5-12，其中场景就是虚拟三维空间，承载所有物体的容器；相机则是控制视线的位置、方向，角度，相机内的画面就是在编程区域所能看到的内容；渲染器代表着渲染的结果使用何种方式绘制在页面的哪个元素上。凡是使用 Three.js 进行 Web 端的 3D 应用开发，必须按照创建场景、设置相机角度、设置渲染器窗口参数这三个流程的顺序才能将预计的场景渲染到页面上去。

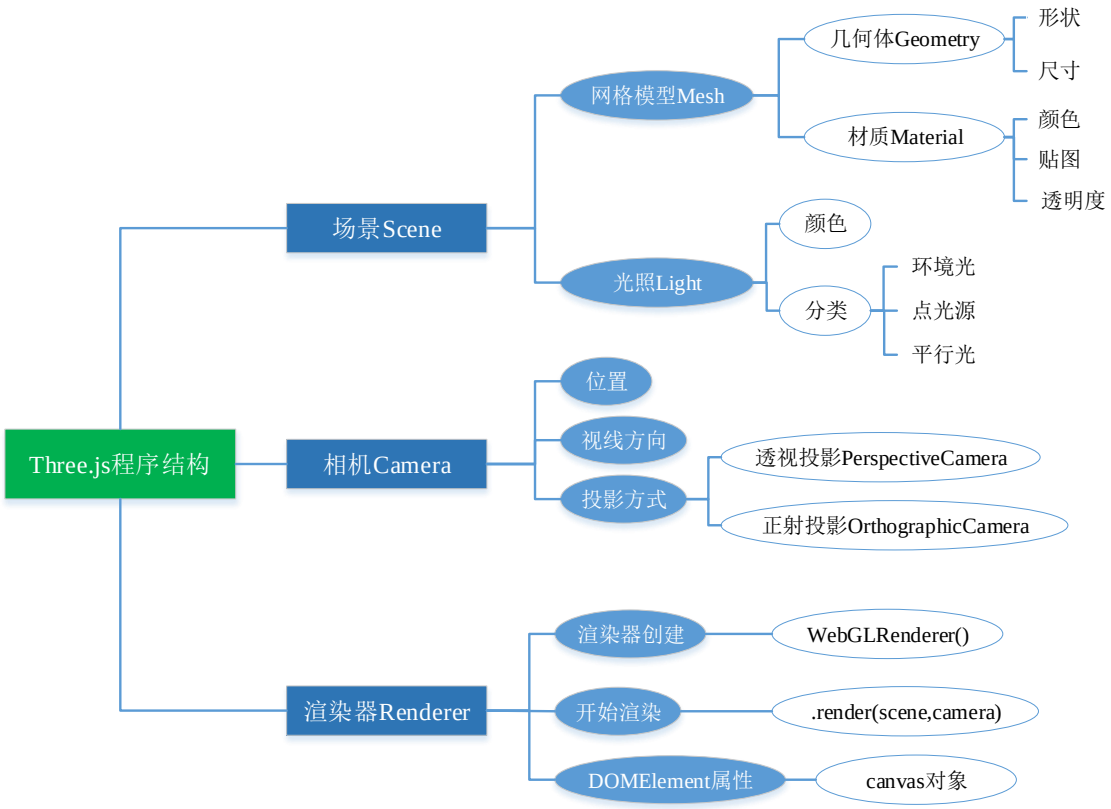


图 5-12 Three.js 程序结构

接下来对系统中的每个模型处理模块进行详细分析。

5.3.1 场景初始化

工作台页面的三维场景是由灯光、相机、背景、坐标轴、底部网格组成，所以在页面渲染前需要对以上部件进行初始化。灯光初始化选择了两个处于Z轴的点光源，颜色为 0xffffff，方向互为反向，此外还加入了一个颜色为 0xffffff 的环境光，具体实现代码为：

```
export const initialLight = () => {
  let directionalLightLeft = new Three.DirectionalLight(0xffffff); //  设置点光源
  directionalLightLeft.position.set(0, 0, 1000);
  scene.add(directionalLightLeft); //  点光源添加到场景中
  let directionalLightRight = new Three.DirectionalLight(0xffffff); //  设置点光源
  directionalLightRight.position.set(0, 0, -1000);
  scene.add(directionalLightRight); //  点光源添加到场景中
```

```
    let ambient; // 环境光
    ambient = new Three.AmbientLight(0xffffff);
    scene.add(ambient); // 环境光添加到场景中
}
```

相机观察三维模型场景的原理借鉴了人眼观察物理世界的模式，所以相机类型分为正投影相机（OrthographicCamera）和透视投影相机（PerspectiveCamera），如图 5-13 可以看到正投影下，线段的角度不同投射到投影面上的长度就不同，而透视投影在投影面上的长度不仅与线段的角度有关，还与观察距离有关。因为本系统会涉及

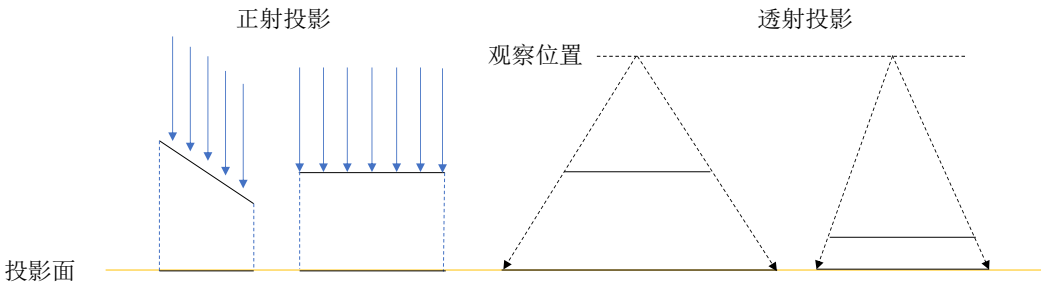


图 5-13 正射投影和投射投影原理

到对物体的放缩处理，如果使用透视投影，当距离太远时会导致物体消失在页面视野中，而正投影是不会出现物体消失在视窗内的现象，因此选择正投影相机作为三维场景的观察器。

背景指的是整个三维场景的背景画面，Three.js 支持纯色和自定义贴图渲染，因为背景不是系统的主要展示区域，所以画布背景设置为 0x333333 的纯色。

坐标轴初始化时是为了让模型导入后给用户一个可参考的坐标系，方便用户查看模型。具体绘制算法为：先定义三个分别指向 X、Y、Z 轴的向量 dirX、dirY、dirZ，将它们的长度规定范化为 1，再创建 Three. ArrowHelper 箭头对象，并设置三个方向的颜色和长度，然后将三个向量传入箭头对象中，实例化三个指向 XYZ 轴的箭头，最后将其加入 Scene 对象中。

底部网格也是用来给用户提供一个模型范围参考区域，具体绘制算法为：根据模型尺寸设定网格矩形区域的长宽，在 X 轴和 Y 轴方向各定义两条 Line 对象，然后等距复制这两个 Line 对象，直至参考区域被横竖线段填充完毕，最后将若干条 Line 对

象加入 Scene 对象中。

当初始化的对象都被添加进整个场景中，就要开始执行渲染，先通过 Three.WebGLRenderer 来创建渲染器对象，再通过渲染器对象暴露出来的 render 函数执行渲染操作，最终初始化效果如图 5-14。

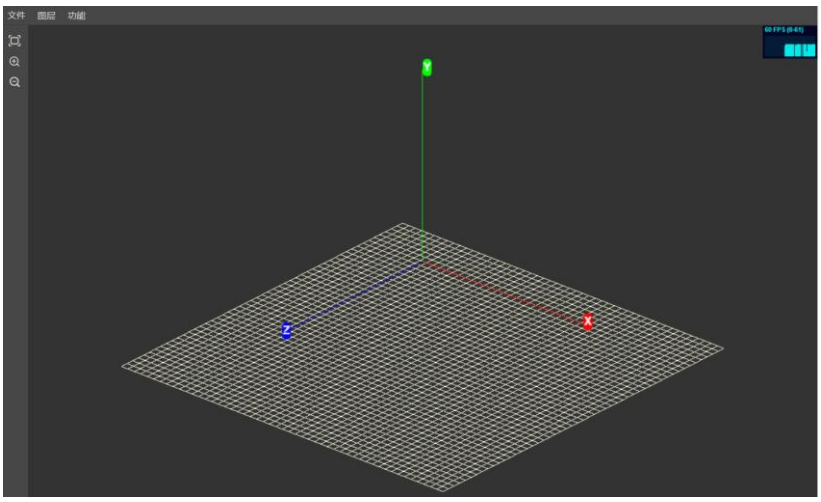


图 5-14 三维场景初始化效果

5.3.2 模型加载与导出

1. 模型加载：用户在我的模型库页面选择符合条件的模型数据进入工作台页面，前端会根据选择的模型资源地址去向服务器发送请求，访问服务器上的文件资源，服务器接收到请求后将数据返回给前端，前端接收到服务器返回的二进制流数据，再同步使用 Three.js 提供的 STLLoader 处理函数将 ArrayBuffer 格式的数据解析成便于 WebGL 渲染的 JSON 格式数据，模型导入过程如图 5-15，模型导入效果如图 5-16。

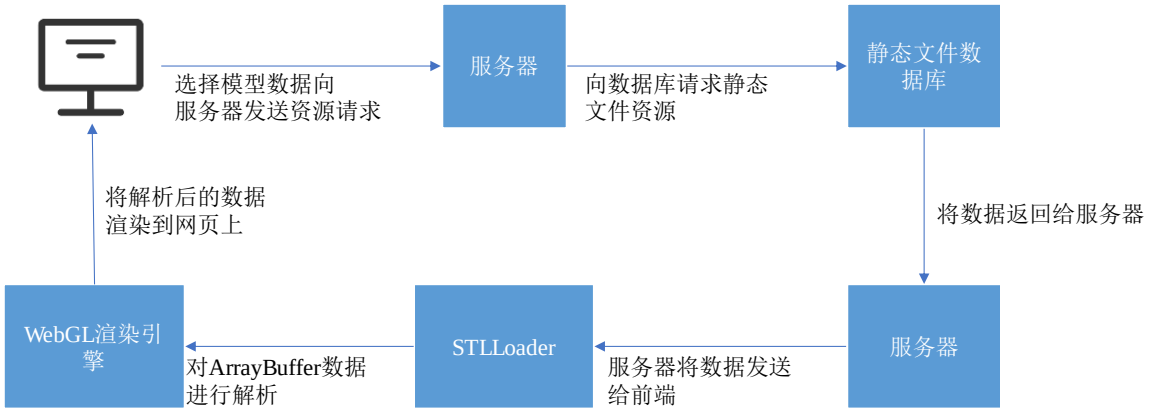


图 5-15 模型导入实现过程

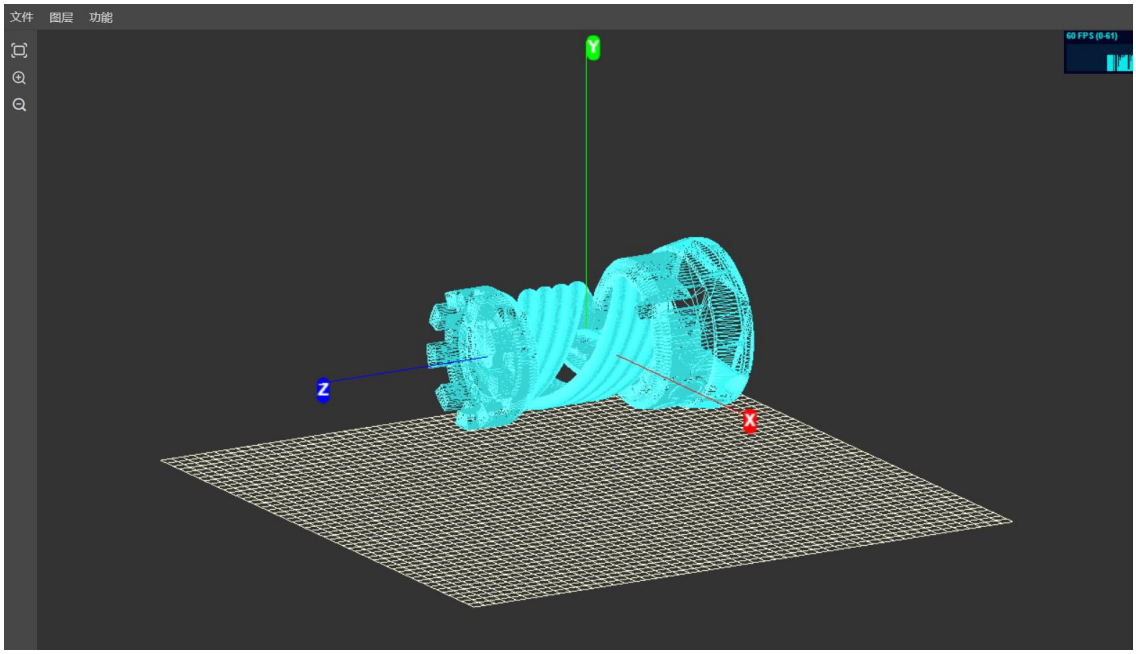


图 5-16 模型导入效果图

2. 模型数据导出：通过遍历当前 Scene 对象将页面上 Visible 属性为 True 的对象添加进新创建的 Scene 对象中，再把新的 Scene 对象数据转化为 Blob 二进制格式使用 STLExporter 函数将其导出，效果如图 5-17。

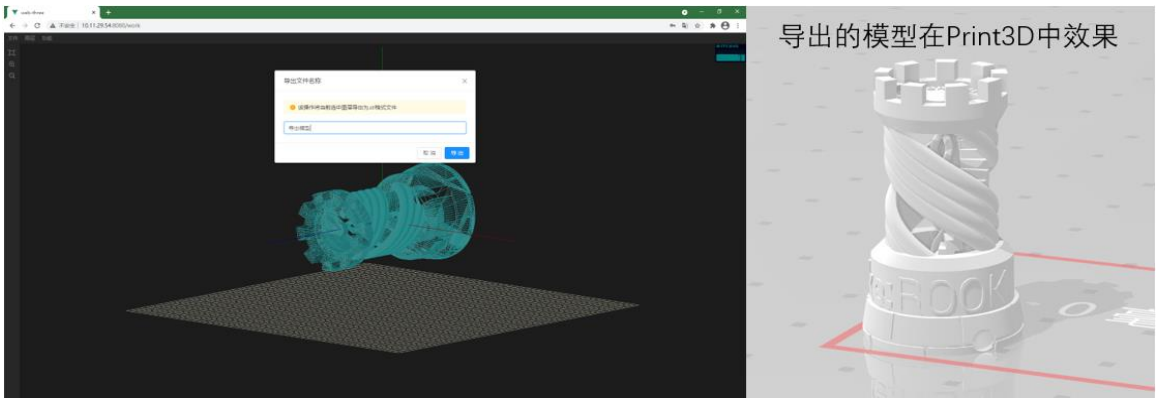
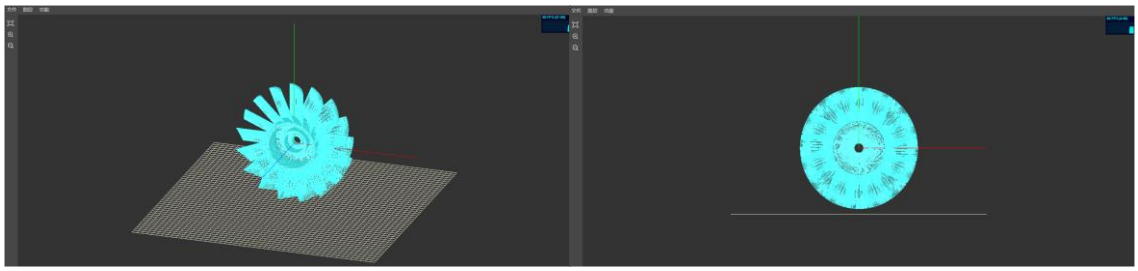


图 5-17 模型导出效果示意图

5.3.3 场景基本交互操作

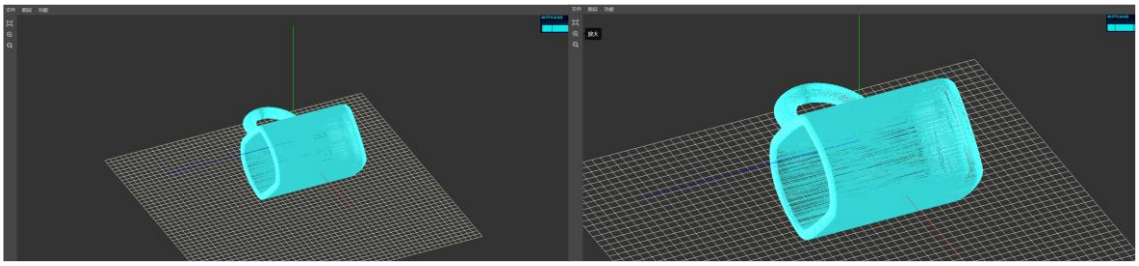
1. 还原：模型在操作过程中会被鼠标旋转到各个角度，此时用户需要模型一键回正功能，将视窗角度重新调整到初始位置。具体实现为，设置窗口宽高比，再设置视野深度本文默认为 1000，调整相机的投影矩阵，设置相机位置，最后执行 updateProjectionMatrix 函数更新相机投影矩阵，效果如图 5-18。



（a）旋转后　　（b）还原

图 5-18 视窗还原

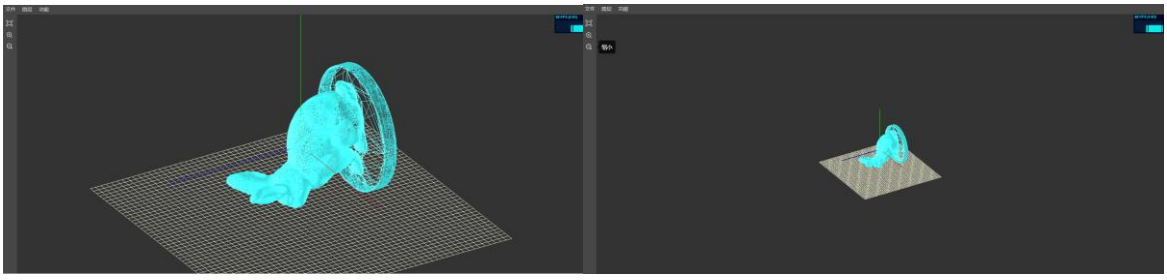
2. 放大：放大页面需要对投影矩阵中的深度系数 S 进行调整，本文放大系数为 0.9，每次点击放大按钮时将页面深度 S 赋值为 0.9S，然后执行 updateProjectionMatrix 函数更新相机投影矩阵，完成放大功能，效果如图 5-19。



（a）原始　　（b）放大

图 5-19 视窗放大

3. 缩小：缩小页面与放大页面同理，只需将系数 S 赋值为 1.1S 即可，效果如图 5-20。



（a）原始　　（b）缩小

图 5-20 视窗缩小

5.4 数据库实现

本文数据库使用非关系型数据库 MongoDB，开发中使用 Mongoose 框架来进行实现。系统所涉及到的数据集合有用户信息和模型信息两类，Mongoose 中是使用

Scheme 结构去定义数据模型，下面是用户信息 Scheme 与模型数据 Scheme 的详细设计。

5.4.1 用户信息 Scheme

用户信息集合需要存储用户名、真实姓名、密码、年龄、单位、专业方向、可用模型索引、用户角色、用户注册时间这些字段，详细数据类型如表 5-5。

表 5-5 用户信息 Scheme

字段	参数	类型
用户名	username	{type: String, default: 'user'}
真实姓名	realname	{type: String, default: '暂无'}
密码	password	{type: String, default: '123456'}
年龄	age	{type: Number, default: 0}
单位	company	{type: String, default: '暂无'}
专业	major	{type: String, default: '暂无'}
可用模型	usableModel	{type: mongoose.Schema.Types.ObjectId}
角色	role	{type: String, default: 'user' or 'admin'}
注册时间	date	{type: Date, default: Date.now}

5.4.2 模型数据 Scheme

模型数据集合需要存储模型描述、模型图片地址、模型文件地址、模型名字、模型所有人 ID、是否公共模型、模型入库时间，详细数据类型如表 5-6。

表 5-6 模型数据 Scheme

字段	参数	类型
模型描述	modelDesc	{type: String, default: '暂无'}
模型图片地址	modelImgUrl	{type: String, default: ''}
模型文件地址	modelFileUrl	{type: String, default: ''}
模型名字	modelTitle	{type: String, default: '暂无'}
模型所有人 ID	ownerId	{type: String, default: ''}
是否公共模型	isPublic	{type: String, default: false}
注册时间	date	{type: Date, default: Date.now}

5.5 服务端实现

服务端使用 Express.js 框架，可以快速搭建小型灵活的后台服务。本系统的中后端服务主要涉及到登录验证、用户信息处理和模型信息处理。

5.5.1 Token 认证

由于前后端分离模式，登录校验和权限接口认证需要依赖 token 机制来实现，本文使用的是基于 JWT 的认证方案，具体实现如下：

1. 用户在登录页面提交用户名和密码；
2. 服务端获取用户名和密码，通过数据库进行查询校验；
3. 服务端生成 Token 返回给前端，本文的 Token 生成规则是 HS256；
4. 前端获取到 Token 后将 Token 保存在 localStorage 中；
5. 之后凡是涉及到权限的请求接口前端会将 Token 放在请求的头部中；
6. 后端接收到前端返回的 Token 并进行解码，得到 Header 和 Payload，然后通过密钥来加密生成 Token 以此检验当前 Token 是否有效。

基于 JWT 的 Token 认证实现如图 5-21。

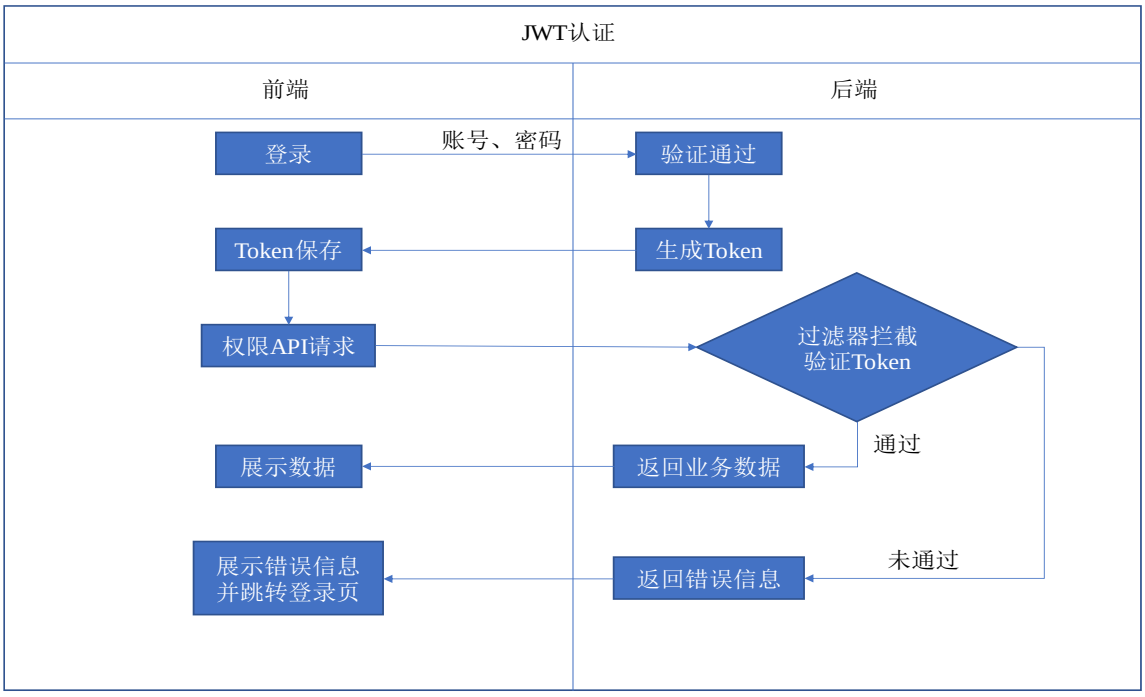


图 5-21 基于 JWT 的 Token 认证实现示意图

5.5.2 用户信息接口服务

前端对用户信息进行操作，需要服务端提供以下接口：登录接口、注册接口、获取用户信息接口、更新用户信息接口、获取用户列表接口、删除用户接口。详细接口设计如表 5-7。

表 5-7 用户信息接口服务详情

接口说明	字段	参数
登录接口	/tokens/login	用户名、密码
注册接口	/tokens/register	用户名、真实姓名、密码
获取用户信息接口	/tokens/userInfo	用户 ID
更新用户信息接口	/tokens/updateUserInfo	用户 ID、用户信息
获取用户列表接口	/tokens/userList	用户名 OR 真实姓名
删除用户接口	/tokens/deleteUser	用户 ID

5.5.3 模型数据接口服务

模型数据信息操作需要服务端提供下列接口：上传文件接口、获取模型列表接口、新增模型接口、删除模型接口、更新模型接口、公共模型使用接口、模型移除可使用接口。详细接口设计如表 5-8。

表 5-8 模型数据接口服务详情

接口说明	字段	参数
上传文件接口	/model/upload	用户 ID、文件数据
获取模型列表接口	/model/getModelList	用户 ID
新增模型接口	/model/userInfo	用户 ID、模型信息
删除模型接口	/model/deleteModel	用户 ID、模型 ID
更新模型接口	/model/updateModel	模型 ID、模型信息
公共模型使用接口	/model/addUsableModel	模型 ID、用户 ID
模型移除可使用接口	/model/removeUsableModel	模型 ID、用户 ID

5.6 系统测试

5.6.1 服务器部署

基于 Web 的增材制造预处理系统的服务器部署，本文使用 UCloud 云主机，型号是 AMD 版，系统盘为 40G RSSD 云盘，BGP 带宽为 1M。服务器部署分为前端、后端、数据库三部分。前端部署：前端将访问后端的地址修改为服务器的公网 IP，再通过 npm run build 命令将开发环境代码打包成生产环境代码，然后将生产环境代码部署在服务器的 8080 端口上；后端则将后端代码打包推送到云服务器上，再通过 pm2 工具启动后端服务；数据库则是在服务器上安装 MongoDB 并配置数据存储路径，再将本地数据服务器启动。本文的云服务器公网 IP 为 106.75.190.235，因此通过网址 http://106.75.190.235:8080 即可访问应用。

5.6.2 功能测试

1. 登录系统：用户名 myTest，真实姓名测试号，密码 123456.

图 5-22 登录系统

2. 上传模型：模型名称螺旋桨，模型描述螺旋桨，设置权限公开，上传图片螺旋桨.png，上传模型螺旋桨.stl.

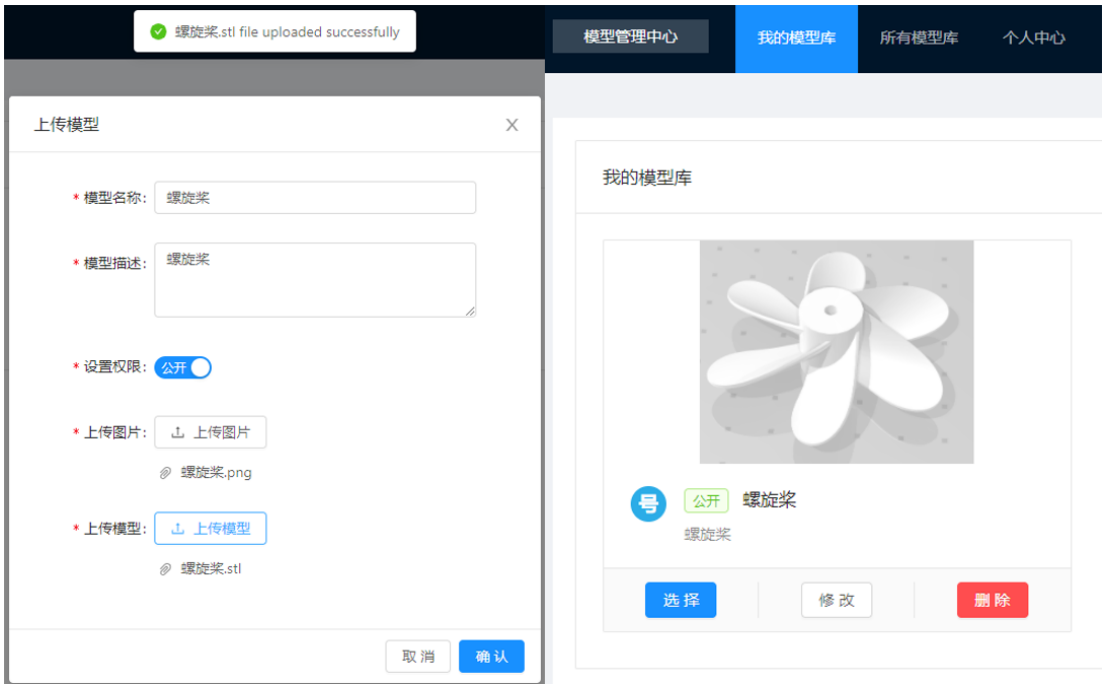


图 5-23 上传模型

3. 修改模型：模型描述修改为螺旋桨修改，设置权限修改为私有。

图 5-24 修改模型

4. 所有模型库中公共模型加入我的模型库。

图 5-25 公共模型加入我的模型库

5. 个人中心。

图 5-26 个人中心

6. 选择模型进入工作台页面。

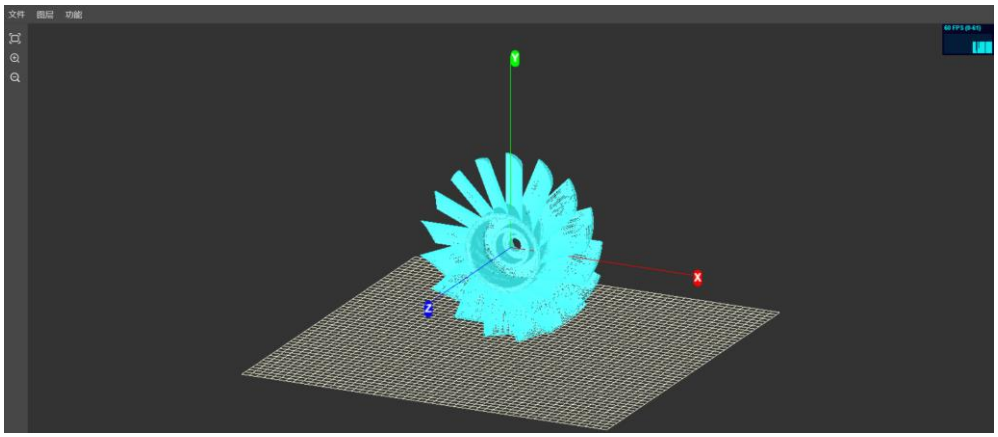


图 5-27 模型进入工作台页面

7. 分层切片：起始切片高度-5，终止切片高度 5，层厚 1，绘制颜色 0xff0000.

图 5-28 分层切片

8. 轨迹生成：轨迹密度为 1，绘制颜色为 0xffffff.

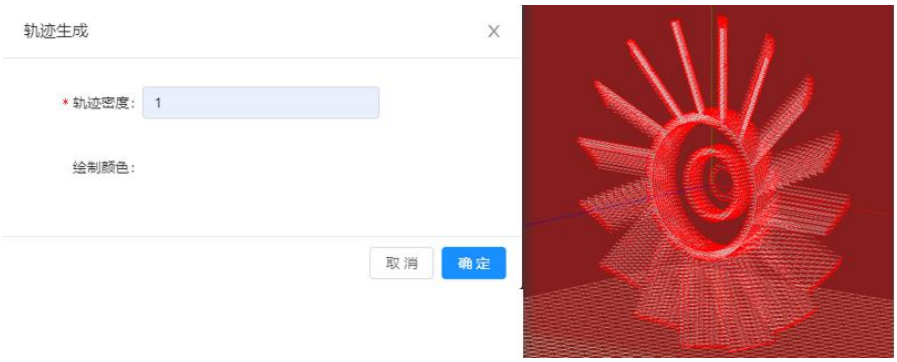


图 5-29 轨迹生成

9. G 代码生成与下载。

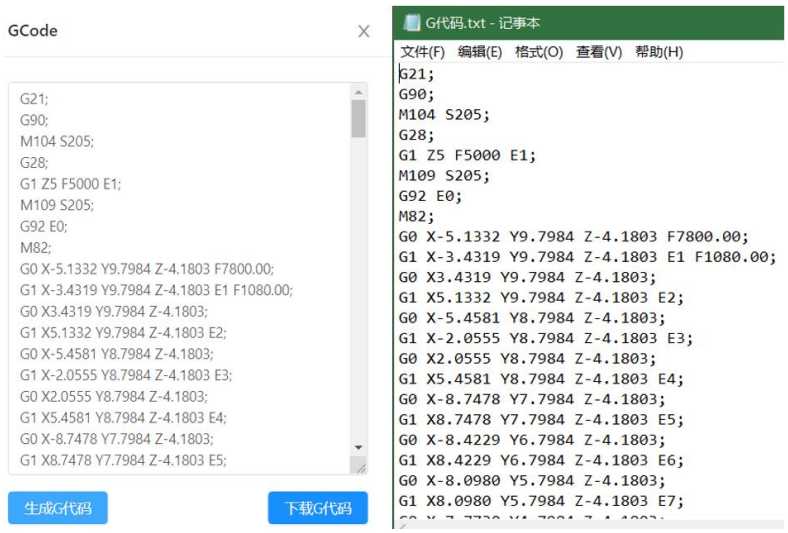
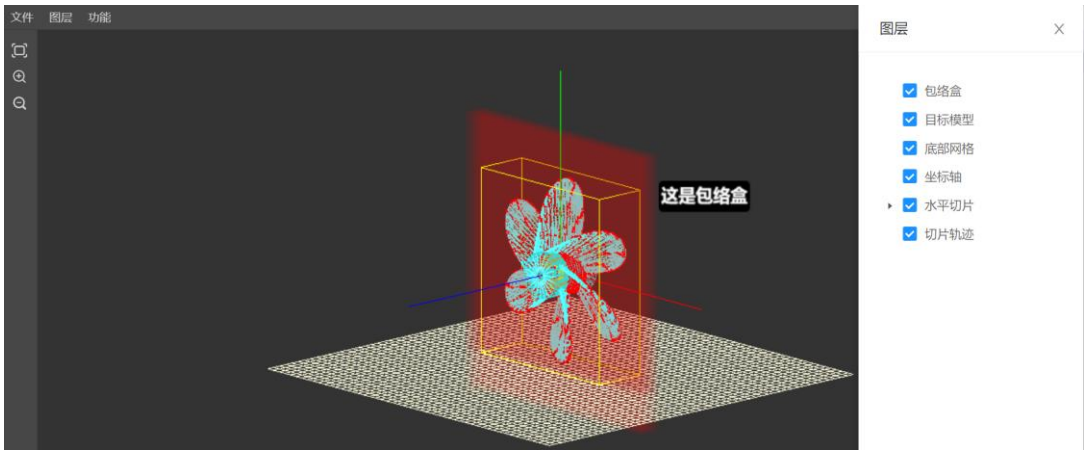
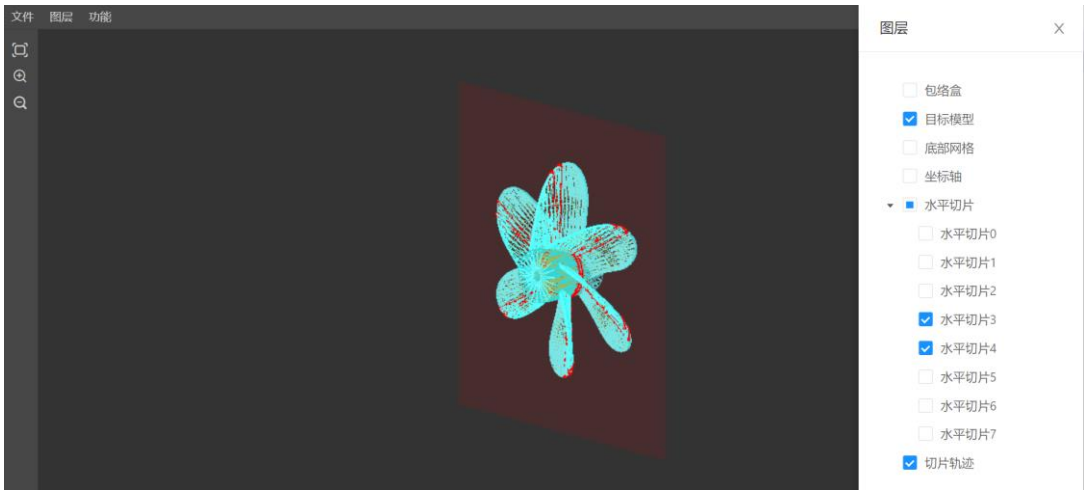


图 5-30 G 代码生成与下载

10. 图层显示与隐藏。



（a）图层全显示



（b）图层部分隐藏

图 5-31 图层显示与隐藏

11. 动画模拟轨迹

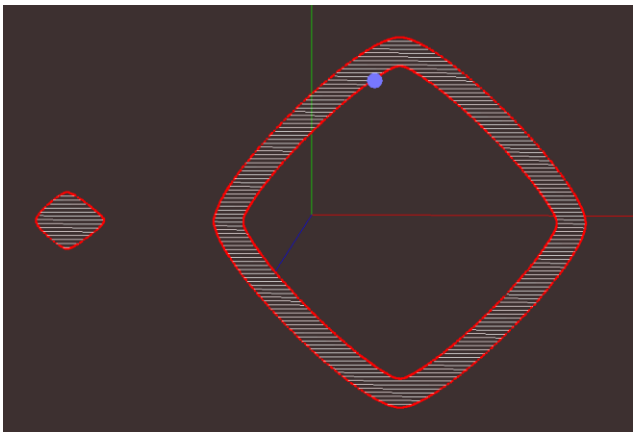


图 5-32 动画模拟填充轨迹

12. 日志记录。

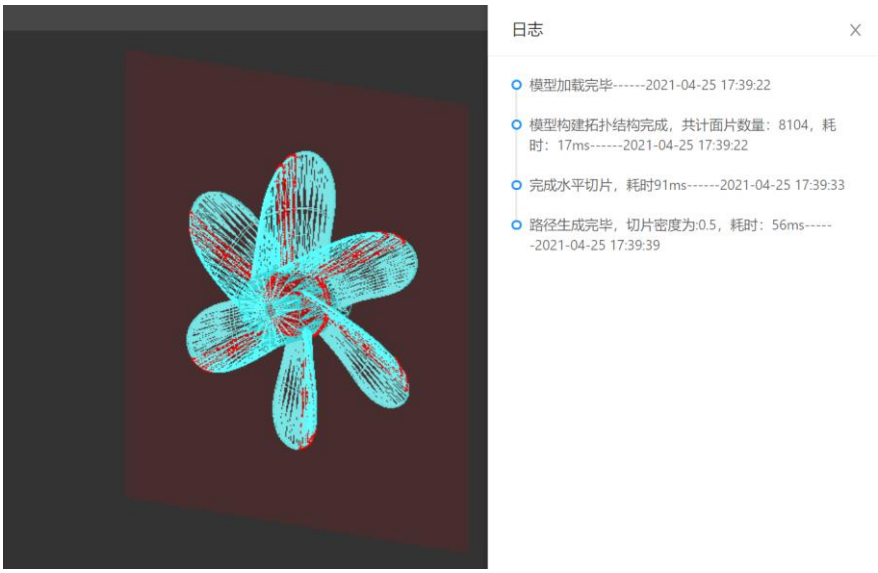


图 5-33 日志记录

13. 用户信息管理。

图 5-34 用户信息管理页面

14. 模型信息管理。

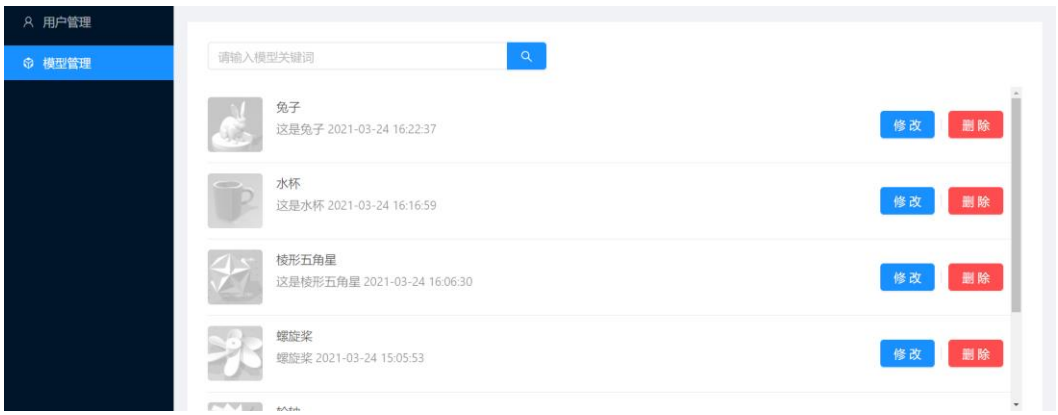


图 5-35 模型信息管理

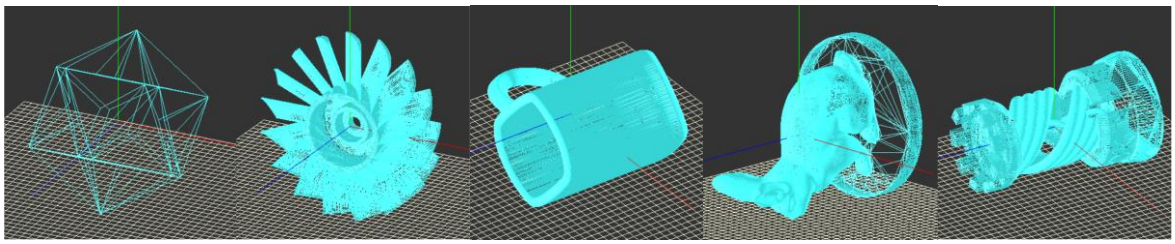
在功能测试中，针对系统的 6 个模块编制了功能测试用例表，如表 5-9，并对每个模块中的功能项进行实际测试。

表 5-9 系统功能测试内容表

模块	测试内容	实际结果
注册登录模块	注册系统账号并登录系统	符合预期
用户模型管理模块	用户对模型数据进行上传、修改、删除、共享	符合预期
模型显示交互模块	定义模型展示区域，对视窗进行还原、放缩操作，控制模型数据、参考坐标、包络盒、分层切片、轨迹的图层显隐，还有模拟轨迹动画	符合预期
增材制造预处理模块	对模型数据进行拓扑重建、分层切片、轨迹规划处理	符合预期
系统用户信息管理模块	对系统所有用户信息进行搜索、删除、修改	符合预期
系统模型管理模块	对系统所有模型数据进行搜索、删除、修改	符合预期

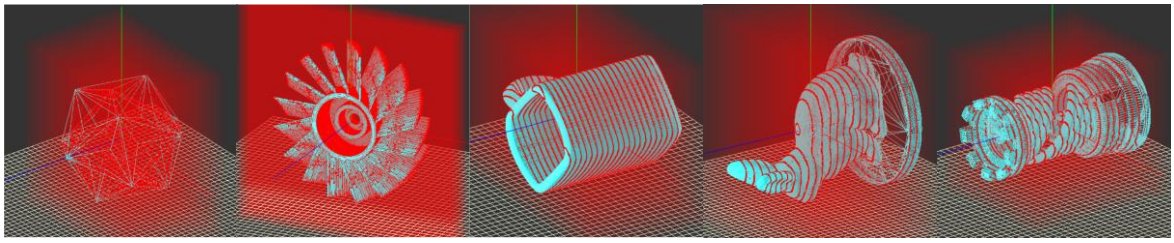
5.6.3 性能测试

性能测试使用的电脑配置为：Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 处理器、8GB RAM 和 NVIDIA GeForce GTX 1050 Ti 显卡，并将系统运行在 Google Chrome 89.0.4389.90 版本上。测试模型使用 5 个文件大小和样貌形态都不同的 STL 模型文件，系统初始形状如图 5-36，分层切片效果如图 5-37，轨迹规划效果如图 5-38。



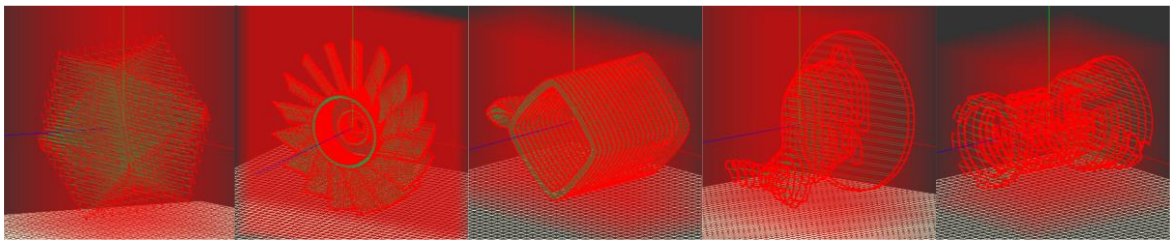
（a）菱形五角星　（b）轮轴　（c）水杯　（d）兔子　（d）旋转城堡

图 5-36 系统测试模型截图



（a）菱形五角星　（b）轮轴　（c）水杯　（d）兔子　（e）旋转城堡

图 5-37 分层切片测试截图



（a）菱形五角星　（b）轮轴　（c）水杯　（d）兔子　（e）旋转城堡

图 5-38 轨迹规划测试截图

本系统主要功能为增材制造预处理流程，需要对每一步的处理算法进行测试。判断在线应用程序是否良好的直观标准是用户等待时间，一般用户可接受的 Web 程序等待时间为 5s，如果某个算法的处理过程用时超过 5s，说明该系统的用户体验有待提高；并且判断在线三维处理应用性能还要参考刷新率指标，实际使用中人眼可接受的刷新率范围在 30fps 到 60fps 之间。下面使用不同的模型来对算法进行测试，拓扑重建、分层切片、轨迹填充的测试结果如表 5-10、表 5-11、表 5-12。

表 5-10 拓扑重建算法测试

拓扑重建算法			
模型名称	面片数量(个)	刷新率(FPS)	实际用时(MS)
棱形五角星	60	60	6
轮轴	19898	60	154
水杯	68730	56	338
兔子	75710	52	457
旋转城堡	219828	50	1118

表 5-11 分层切片算法测试

分层切片算法	分层密度为 1		
模型名称	分层数量(层)	刷新率(FPS)	实际用时(MS)
棱形五角星	26	60	188
轮轴	20	51	362
水杯	100	45	2772
兔子	20	54	239
旋转城堡	24	55	1148

表 5-12 轨迹填充算法测试

轨迹填充算法	分层密度为 1	填充密度为 1	
模型名称	轮廓数量(个)	刷新率(FPS)	实际用时(MS)
棱形五角星	26	60	57
轮轴	60	49	648
水杯	236	40	2543
兔子	31	56	46
旋转城堡	83	53	83

通过上面的测试结果得知，随着面片数量从 60 片增长到 21 万片时，算法耗时从 6ms 增长到 2700ms，但都在用户可接受范围之内，并且通过调研显示大部分用于增材制造处理的模型文件数据量都在 10 万面片以下，并且用户的机器配置越高，那么系统的流畅度就会越高，因此本系统可以满足增材制造预处理的算法需求。

系统高并发测试采用 Python 的多线程去模拟并发效果，向服务器施加并发访问压力，检测系统在并发压力下响应接口的能力。在 Python 并发测试代码中计算错误请求个数以及 HTTP 平均响应时长，同时使用 pm2 提供的 monitor 性能监视器来监听 CPU 占用率等情况。

高并发测试的模拟请求为 1000 个，系统只需在 1000 个请求压力情况下还能较为流畅的进行功能的使用且服务器主机的 CPU 占用率不超过 50%，就说明本系统满足实际运行场景需求。本文将初始并发访问数量设置为 200 个，每间隔 10 秒增加 200 个并发用户，直到模拟 1000 个并发用户数为止，同时对服务器的各项性能进行监听。上述测试结果经整理如表 5-13。

表 5-13 并发测试结果

并发数	错误请求数(个)	HTTP 平均响应时长(MS)	CPU 占用率(%)
200	0	11ms	31
400	0	12ms	30
600	0	13ms	34
800	0	14ms	32
1000	0	12ms	35

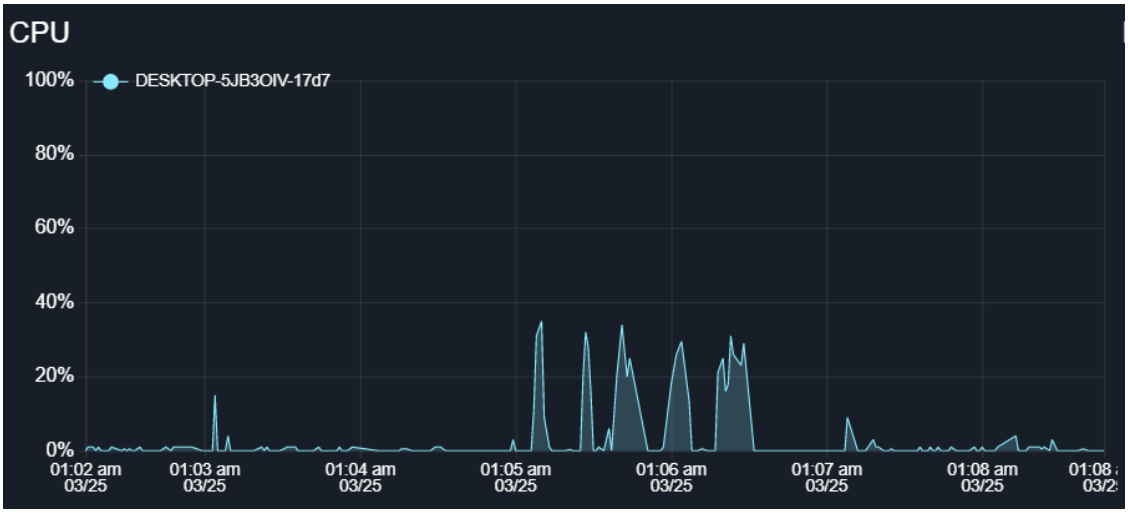


图 5-39 并发测试 CPU 占用率

根据图 5-39 的测试结果显示，系统内请求的平均响应时间为 12.4ms 左右，随着并发数量从 200 个递增到 1000 个，CPU 占用率也从 31%递增至 35%。由实验结果得知服务器在 1000 个并发访问压力下，只要网络环境没有太大波动，HTTP 请求的响应时间也不会有太大变化，并且随着并发量的上升，服务器主机的 CPU 使用率只是稍微增加，完全可以满足实际生产需求。

5.7 本章小结

本章对增材制造预处理 Web 平台的实现进行了详细的阐述和展示，并对系统进行功能测试和性能测试，从测试结果来看，系统满足预期的开发要求。

6. 总结与展望

6.1 本文主要内容及结论

本文主要内容主要分为两大部分：

1. 将增材制造预处理相关软件的算法迁移至 Web 端，把算法在原有的基础上根据 JavaScript 语言的数据结构特性进行改进，使算法满足在 Web 端实时处理的要求，并结合近几年较为热门的 WebGL 技术，流畅美观地实现模型数据的展示与交互。

2. 严格按照前后端分离原则，把用户管理、模型管理与增材制造模型数据预处理功能整合到一个 Web 应用中，并开发出一个即时运行、便捷高效、数据可控的增材制造预处理 Web 系统。

首先针对 JavaScript 语言的内置数据结构特性设计了增材制造预处理的几个核心模块算法，主要包括 STL 模型的导入导出、拓扑重建、分层切片、轨迹填充和 G 代码生成，并将这些模块封装成黑盒函数供程序使用；再调用 WebGL 图形接口，使用二次封装的 Three.js 框架完成模型数据的显示与交互操作，针对一些实际业务需求，比如放缩还原，坐标轴和底部网格的绘制，场景图层的显示与隐藏等，自行设计算法实现；然后基于前后端分离的开发模式去对实现整个系统功能，包括前端页面、后端接口服务和数据库集合存储等；最后对 Web 平台进行了相关测试。

由实验结果表明：本文开发基于 Web 的增材制造预处理系统用户体验良好，功能实现完整，算法准确可行，应用界面美观，基本可以满足增材制造数据预处理的需求。

6.2 展望

由于增材制造预处理软件大多数是客户端，开发成本高，无法跨平台还容易被破解，随着增材制造技术的普及，未来相关应用最好是能跨平台，即开即用，且要有一定的数据管理能力。Web 应用正好可以满足上述需求，并且网页的三维开发技术变

得越来越成熟，在此背景下，本文是对增材制造数据预处理结合 Web 应用的一个探索实践，距离真正成熟的商业化系统，仍有以下几点改进方面：

1. 由于系统研究的对象是 STL 模型数据，只会关注模型是否按照流程被完整处理，而在实际打印过程中需要考虑打印的工艺，可能会添加支撑元素，或使用其它的切片方式，这都需要跟进开发相应的处理算法。

2. 系统交互方式有待丰富，暂时很多场景的背景灯光、模型材质和渲染颜色都是固定的，如果需要更自由的展现方式，还需要针对每个场景中的对象进行定制化开发。

3. 目前系统的处理模型只是中小模型，如果涉及大型模型数据，还要对算法再进行优化，并且可以考虑使用多个服务器进行同步计算，达到计算资源均衡的目的。甚至还能探索新系统架构去实现均衡计算，例如华为研发的鸿蒙系统就实现全场景分布式计算。

4. 由于系统带有数据管理性质，具备一定用户黏性，可以考虑加入更多实用功能，例如模型数据的版权认证，用户团队创建，移动端适配等。

5. 2017 年推出的 WebAssembly 技术可以让浏览器直接执行其他高级语言，比如 C、C++、Go 语言等，旨在解决高计算网页应用的问题。目前四大浏览器厂商和 W3C 联盟正在撰写 WebAssembly/GC 标准，因此未来可以结合 WebAssembly 对系统进行改进，增加实时处理的流畅度。

致 谢

几砚昔年游，于今已二秋，随着两年韶华匆匆逝去，我的研究生生活也将画上句号。在这段纯粹而浪漫的时光里，我变得更加成熟稳重，不仅明白将来需要背负的责任，也找寻到了属于我的人生意义，虽然过程不乏跌宕起伏，但人生不就是在波澜之中才壮阔起来的吗。因此，我要特别感谢这些可爱的人。

首先，我要感谢我的导师——李国宽老师，平日除了给我的科研方向提供建议，还时常关心我的生活，在论文修改期间也会从繁忙的工作中抽出时间耐心与我讨论问题，我从中受益良多。

其次，我要感谢张海鸥教授和王桂兰教授，他们的团队在增材制造领域取得十分卓越的成绩，团队内部学风优良，组员之间相处融洽；我在组里不仅学习到了丰富的知识，也学到许多为人处世的道理。此外，特别感谢组内的戴福生博士，论文思路是在戴博的启发下确定的，并且对于不懂的问题，戴博会非常仔细地给我讲解，让我少走了很多弯路。

然后我还要感谢武汉光电国家研究中心研会主席团的小伙伴们，在为学院的各大活动群策群力时，我既锻炼了自己的综合素质能力，又收获到了许多珍贵的友情，让我的科研之路不再孤单。

此外，我还要感谢武汉理工国标舞协会以及指导老师朱雄灏老师，这是我在本科创立的社团，每周固定的训练与多次参赛经历在增强我技术水平的同时，也提高了艺术审美。朱雄灏老师倾囊相授，带我见识到许多顶尖舞者的魅力，开拓人生更多的可能性。

最后，我要真心感谢与祝福我的家人。感谢父母，感谢外公外婆，焉得谖草，言树之背，养育之恩，无以为报；祝福弟弟，马上你也要高考了，希望可以不负众望，一举夺魁；感谢舅舅舅妈，在求学生涯中给予我生活上很大的帮助；感谢女朋友许愿，是你陪伴我度过研究生最美好的时光，希望我们会一起创造出更多精彩的未来。

生年不满百，常怀千岁忧。昼短苦夜长，何不秉烛游!

参考文献

1. 胡浩亮，陈萍，张争艳，陶孟仑，陈定方，陈琼. 基于 AutoCAD 的三维 CAD 模型直接切片方法. 湖北工业大学学报，2014(4)：73-75
2. Chen Q G, Zhang J C. Design and Realization of Auto-Programming System for SLS on CAD. Advanced Materials Research, 2013, 662:879-883
3. 谢明师. 3D 打印预处理软件设计与实现[硕士学位论文]. 太原：中北大学，2017
4. 卢秉恒，李涤尘. 增材制造(3D 打印)技术发展. 机械制造与自动化，2013，42(04)：1-4
5. Caffrey T. Additive manufacturing and 3D printing state of the industry annual worldwide progress report. engineering management research, 2013, 2(1):209-222
6. 陈志茹，夏承东，李龙，楚瑞坤，周德敬. 3D 打印材料. 金属世界，2018(05):15-18
7. Gordeev E G, Galushko A S, Ananikov V P. Improvement of quality of 3D printed objects by elimination of microscopic structural defects in fused deposition modeling. PloS one, 2018, 13(6): e0198370
8. Tosto C, Saitta L, Pergolizzi E, Patti A, Cicala G. Fused Deposition Modelling (FDM): New Standards for Mechanical Characterization. Macromolecular Symposia, 2021, 395(1):2000253
9. Ruan M. The Survey of Vision-based 3D Modeling Techniques. Journal of Physics Conference Series, 2017, 910(1):012012
10. 谢越. 面向普通用户的三维模型设计方法研究[博士学位论文]. 杭州：浙江大学，2016
11. Eracar Y A, Kokar M M. Using UML and OCL for representing multiobjective combinatorial optimization problems. Journal of Intelligent Manufacturing, 2014, 25(3):555-569
12. Y Zhang, Z Liu, X Wei, X Li, Q Zhang. Generative recursive network for constructive solid geometry. Electronics Letters, 2019, 55(14): 785-787

13. Requicha A A, Voelcker H B. Solid modeling: Current status and research directions. Computer Graphics and Applications, IEEE, 1983, 3(7):25-37
14. Requicha A A, Voelcker H B. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. Proceedings of the IEEE, 1985, 73(1):30-44
15. Braid I C. The synthesis of solids bounded by many faces. Communications of the ACM, 1975, 18(4):209-216
16. Xinyao L I, Zhang W, Chen L. A fast analytic method for CAD model based on boundary representation in fixed grid. Acta Aeronautica et Astronautica Sinica, 2019
17. Braid I, Hillyard R. Geometric modeling in ALGOL 68. ACM Sigplan Notices, 1977, 12(6):168-174
18. 阿占文. 家庭化 3D 打印机分层软件的设计与实现[硕士学位论文]. 武汉：华中科技大学，2014
19. 牛燃恒. 电弧 3D 打印机控制系统设计与切片算法研究[硕士学位论文]. 重庆：重庆大学，2019
20. Sketchfab. The leading platform for 3D & AR on the web[EB/OL]. 2021, https://sketchfab.com
21. 动动三维．全球首款 3D/AR 交互内容在线工具[EB/OL]. 2021, https://www.ddd.online
22. 琢刻. 3D 专属内容展示平台[EB/OL]. 2017, https://gizmohub.com
23. Neil Savage. Weaving the web. Communications of the ACM, 2017, 60(6): 20-22
24. Tabarés Raúl. HTML5 and the evolution of HTML; tracing the origins of digital platforms. Technology in Society, 2021, 65:101529
25. 孟秀霞. Web UI 技术的研究与应用[硕士学位论文]. 北京：中国地质大学，2019
26. Claussnitzer, Melina, Cho, Judy H, Collins, Rory, et al. A brief history of human disease genetics. Nature, 2020, 577(7789), 179–189
27. Richards G, Lebresne S, Burg B, Vitek J. An analysis of the dynamic behavior of JavaScript programs. Acm Sigplan Notices, 2010, 45(6):1-12
28. Kleber A, Edler D, Dickmann F. Cartography and the sea: A javascript-based web mapping application for managing maritime shipping. The first edition. Wiesbaden: Springer VS, 2020. 173-186

29. Jesse James Garrett. Ajax: A new approach to web applications[EB/OL]. 2005, https://www.adaptivepath.com/publications/essays/archives/000385print.php
30. Stian Reimers, Neil Stewart. Adobe Flash as a medium for online experimentation: A test of reaction time measurement capabilities. 2007, 39(3), 365–370
31. 陆凌牛. HTML5 与 CSS3 权威指南. 第三版. 北京：机械工业出版社，2011
32. Gutiérrez R T. Understanding the role of digital commons in the web; The making of HTML5. Telematics and Informatics, 2018, 35(5): 1438-1449
33. 姚敦红. jQuery 程序设计基础教程. 第一版. 北京：人民邮电出版社，2013
34. Tim Hesterberg. Bootstrap, 2011, 3(6), 497–526
35. Chen C, Su T, Meng G, Xing Z, Yang L. From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, May 2018, Proceedings of the IEEE, 2018: 665-676
36. Nelson R, Shukla A, Smith C. Web Browser Forensics in Google Chrome, Mozilla Firefox, and the Tor Browser Bundle. The first edition. Cham: Springer, 2020
37. Deoptimization pattern analysis and improvement in JavaScript engine. Electronic Technology, 2017, 46(07): 17-23
38. Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 2010, 14(6):80-83
39. 陈瑶. 基于 Node.js 高并发 web 系统的研究与应用[硕士学位论文]. 成都：电子科技大学，2014
40. IOANNIS K, KYRIAKOS-IOANNIS D, NIKOLAOS D. Is Node.js a viable option for building modern web applications? A performance evaluation study. Computing, 2015, Vol.97(10): 1023-1044
41. Abdalkareem R, Oda V, Mujahid S, Shihab E. On the impact of using trivial packages: an empirical case study on npm and PyPI. Empirical Software Engineering, 2020, 25(2):1168-1204
42. Arcos-Medina G, Menéndez J, Vallejo J. Comparative Study of Performance and Productivity of MVC and MVVM design patterns. KnE Engineering, 2018: 241-252
43. 乔淑夷. 基于 MVC 模式的 Web 前端框架关键技术研究与实现[硕士学位论文].

青岛：中国海洋大学，2014

44. 杜艳美, 黄晓芳. 面向企业级 web 应用的前后端分离开发模式及实践. 西南科技大学学报, 2018, v. 33; No.130(02):86-90
45. Barladian B K, Shapiro L Z, Mallachiev K A, Khoroshilov A V, Koverninskii I V. Visualization Component for the Aircraft Real-Time Operating System JetOS. Programming and Computer Software, 2020, 46(3): 167-175
46. 刘璐. 基于 WebGL 与动画技术的滑坡与泥石流演变过程展示技术研究[硕士学位论文]. 北京：北京交通大学，2020
47. 侯聪聪，南琳，张磊. 基于分组的 STL 模型快速切片算法. 制造业自动化，2014，(9): 12-15
48. 巢海远，刘景，童晶，张洛声. 一种处理带有边界的非封闭 STL 模型的切片算法. 计算机集成制造系统，2015，21(10): 2587-2595
49. Gan W, Zhou Y H. An improved method of hash table based on transform and conquer. Information and Computational Science, 2012, 9(17): 5361－5371
50. Hao J B Fang L, Yang H F. An Improved Boundary Extraction Method of STL Model Based on Edge Curvature Estimation. Computer Modeling and New Technologies, 2014, 18(10):252-258
51. Miao Y, Song X, Wang J, Lu Z. NC machining verification algorithm based on the STL model. International Journal of Advanced Manufacturing Technology, 2020(12): 1-9
52. 何泳江. 一种工业 CT 切片直接生成 3D 打印 G 代码的方法研究[硕士学位论文]. 重庆：重庆大学，2019
53. 徐文鹏，王伟明，李航，杨周旺，刘秀平，刘利刚. 面向 3D 打印体积极小的拓扑优化技术. 计算机研究与发展，2015，52(1): 38-44
54. 侯章浩，乌日开西·艾依提. 3D 打印的路径规划研究综述. 机床与液压，2016，44(5): 179-182
55. Wang L, Cui B L, Pan H X, Zhao L, Tian Y. Research and implementation of online design open platform based on vue.js. Information Technology and Informatization, 2019(11): 168-170
56. Wang Y, University S. DDMVVM: Domain-Driven MVVM Design Model. Digital

Technology and Application，2018，36(03): 113-115

57. Yuanyi Chen and Chen Yuanyi. Application of Web Program Development Based on Struts Framework. Journal of Physics: Conference Series, 2020, 1648(3): 032193
58. G Lavoué, Chevalier L, Dupont F. Streaming Compressed 3D Data on the Web using JavaScript and WebGL. Neuroimage, 2013, 84(1):265-278
59. Wang Q, Cheng-Peng X U, Xiao-Qiang X I. Design and implementation of learning management system based on node.js. Electronic Design Engineering, 2018, 26(04): 24-28
60. 王海洋. 基于 Node.JS 的 NoSQL 数据库建立及处理方法：中国，CN111488339A[P]. 2020
61. Duggirala S. NewSQL Databases and Scalable In-Memory Analytics. Advances in Computers, 2018, 109:49-76
62. Sangeeta Gupta. Comparative Analysis of Nosql Specimen with Relational Data Store for Big Data in Cloud. International Journal of Distributed and Cloud Computing, 2015, 3(1):17-23